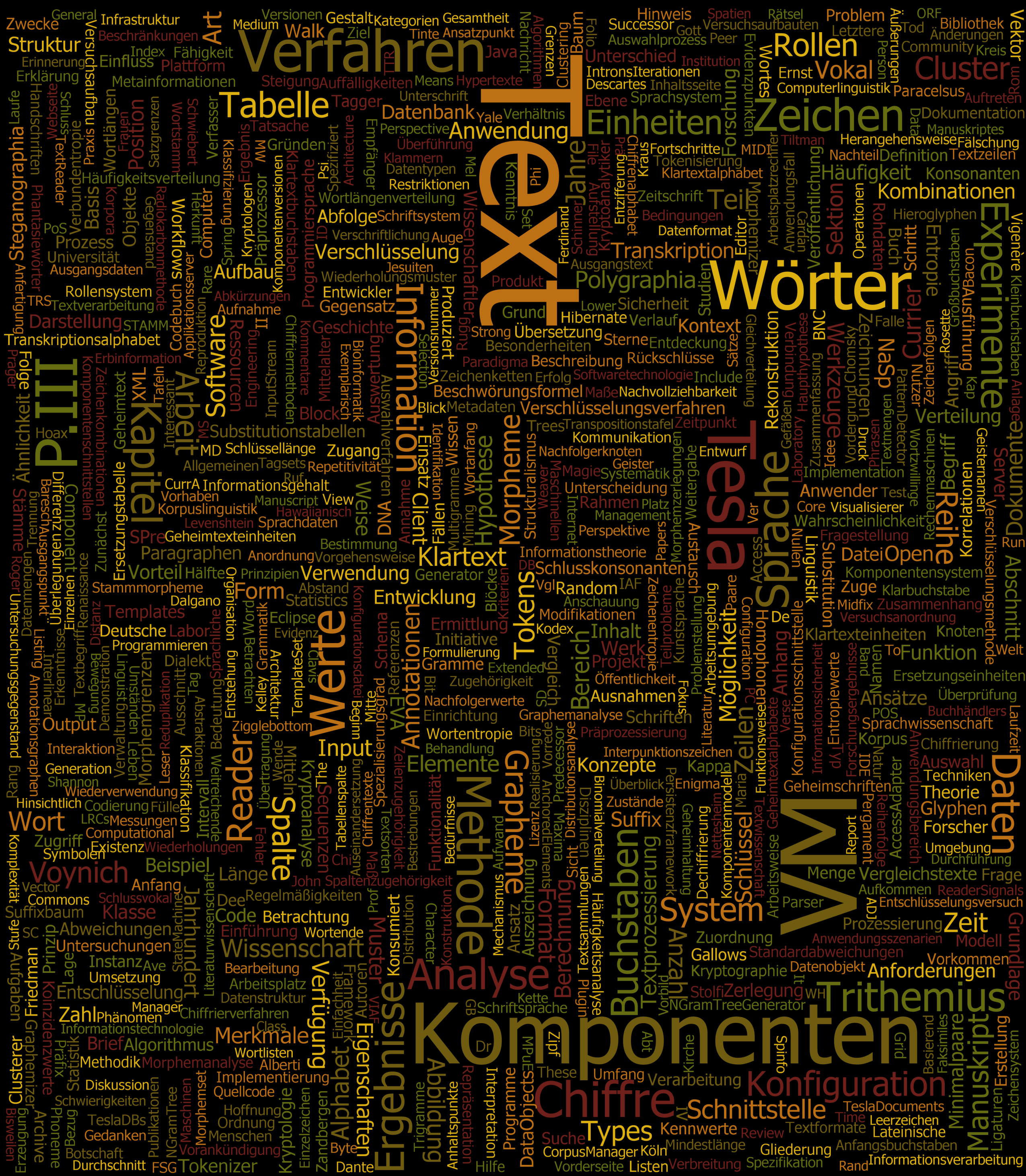


Jürgen Hermes

Textprozessierung

Design und Applikation

Vorwort

Dieses Dokument ist die aktualisierte Fassung meiner Dissertation, die im Wintersemester 2011/2012 von der Philosophischen Fakultät der Universität zu Köln angenommen wurde. Die Referenten waren Prof. Dr. Jürgen Rolshoven und Prof. Dr. Martin Becker. Die Disputation fand am 21. Dezember 2011 statt.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr. Rolshoven dafür, dass ich weitgehende Freiheiten hinsichtlich der Gestaltung und Umsetzung meiner Arbeit hatte und dennoch in vielen Gesprächen auf seinen Rat vertrauen konnte. Außerdem wurden mir durch die Anstellung an seinem Lehrstuhl für Sprachliche Informationsverarbeitung auch die materiellen Sorgen genommen, mit denen viele Promoventen zu kämpfen haben. Das Klima an diesem Lehrstuhl kann wohl als einzigartig bezeichnet werden, ich bin so vielen aktuellen und auch ehemaligen Mitarbeitern zu Dank verpflichtet, dass ich sie hier unmöglich alle aufzählen kann. Mir ist es aber ein Anliegen, zumindest noch die folgenden namentlich zu erwähnen:

An erster Stelle möchte ich Stephan Schwiebert danken, der seine Dissertation zeitlich und thematisch parallel zu der meinen angefertigt hat. Zwei Arbeiten zum gleichen Thema - unserem Text Engineering Software Laboratory - zur gleichen Zeit zu erstellen, ist nicht gerade frei von Schwierigkeiten. Dass wir diese umschiffen und zwei so unterschiedliche Arbeiten hervorbringen konnten, zeugt wohl am besten davon, wie fruchtbar die Art unseres Austauschs war. Wir haben das System Tesla konzeptuell zusammen entworfen, und auch ich habe meinen Beitrag zur Umsetzung geliefert; ohne Zweifel muss aber festgestellt werden, dass ohne die mir bisweilen unheimlichen Programmierkünste meines Kollegen Tesla niemals ein solch zuverlässiges, funktionales und auch ästhetisches Softwaresystem geworden wäre.

Herausragender Dank gilt auch Claes Neuefeind und Christoph Benden, einerseits für die langen Gespräche, aus denen ich eine Fülle von Anregungen für meine Arbeit ziehen konnte und die mich auch vor dem Beschreiten diverser Holzwege bewahrten. Vor allem aber danke ich für ihre gründliche Auseinandersetzung mit dem Resultat meiner Gedanken. Aus dieser Auseinandersetzung ging letztlich dann der Text hervor, der auf

den restlichen Seiten dieser Arbeit zu finden ist. Für die relative Fehlerfreiheit dieses Textes haben neben den beiden letztgenannten auch Frauke Schmidt und Moritz Sommet gesorgt, die ich deswegen in meinen Dank einschließen möchte. Alle verbliebenen Fehler wurden mit an Sicherheit grenzender Wahrscheinlichkeit durch mich selbst im Anschluss an die Korrekturen wieder eingebaut. Auch Sonja Subicin gebührt mein Dank, da sie mir bei vielen Grafiken behilflich war und die durch den Entwurf unzähliger Icons maßgeblich dafür verantwortlich gemacht werden kann, dass Tesla so schön anzusehen ist.

Zum Schluss möchte ich mich auch für die Unterstützung meiner Eltern und Schwiegereltern bedanken, sowie für den Rückhalt, den ich durch meine kleine Familie bekommen habe - Nadine, Lucienne und Julius, euch möchte ich dieses Dokument auch widmen.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung und Überblick

Der Einsatz von Computern in der Forschung ist inzwischen selbstverständlich geworden. Ohne die Fähigkeit moderner Rechenmaschinen zur Speicherung und Bearbeitung großer Datenmengen könnten auf kaum einem Gebiet noch wissenschaftliche Fortschritte erzielt werden. Mit entsprechender Software ausgestattet, können mit Computern Daten gesammelt und editiert werden, sie dienen als Steuerungs- und Kontrollinstanz verschiedenartiger Berechnungen, sie eignen sich zur Simulation komplexer Modelle, sie werden mittels Visualisierung von Zusammenhängen als diagnostische Hilfsmittel eingesetzt. Neben dieser Nutzung als Speicher und Berechner von Datensätzen dienen Computer auch zur Kommunikation über wissenschaftliche Ergebnisse, sowohl für den *Peer to Peer*-Austausch von Forschern untereinander, etwa via e-Mail, als auch über Veröffentlichungskanäle wie Tweets, Blogs, Internet-Achive und Digitale Bibliotheken.

Computer sind nicht nur Spezialisten zugänglich, welche Kenntnis über die auf der Hardware ablaufenden Prozesse haben. Dies liegt daran, dass durch den Einsatz von Software über diese grundlegenden Prozesse abstrahiert werden kann: Betriebssysteme sorgen für die Verwaltung von Speicher- und Ein-/Ausgabemedien sowie für die Steuerung der Programmausführung, darauf aufsetzend bietet spezifische Anwendungssoftware meist über graphische Benutzeroberflächen einen komfortablen Zugriff auf die gewünschten Funktionen. Vor diesem Hintergrund konnte der Computer nicht nur für Computer-Spezialisten (Informatiker), sondern auch für die Mehrheit der Wissenschaftler zum zentralen Arbeitsplatz werden. Dieser Arbeitsplatz ist zumindest teilweise ein virtueller, seine Ausstattung erfolgt zum großen Teil durch spezialisierte Anwendungsprogramme. Die Qualität des Arbeitsplatzes hängt mithin auch davon ab, inwiefern die installierte Software die täglichen Aufgaben des Wissenschaftlers unterstützt. In dieser Arbeit wird der Entwurf, die konkrete Implementation sowie die exemplarische Anwendung einer Software thematisiert, die als Ausstattung eines Arbeitsplatzes für Wissenschaftler, die sich mit der Verarbeitung (Prozessierung) textueller Daten auseinander setzen, dient.

Kapitel 2 beschäftigt sich mit den theoretischen Grundlagen für die Entwicklung eines solchen Arbeitsplatzes: Welche Wissenschaften verarbeiten textuelle Daten? Wie kann daraus ein allgemeiner Textbegriff definiert werden? Welche spezifischen Bedürfnisse, welche Vorzüge ergeben sich durch die Verarbeitung dieses speziellen Datenformats? Welche Ansätze zur Verarbeitung textueller Daten wurden bisher verfolgt? Wie kann man textprozessierende Wissenschaft bestmöglich unterstützen? Aus der Behandlung dieser Fragen werden zum Ende des Kapitels Anforderungen spezifiziert, denen eine Software für den über Texte arbeitenden Wissenschaftler nachkommen sollte.

Kapitel 3 beschreibt ein System, das diese Anforderungen implementiert. Es handelt sich hierbei um das *Text Engineering Software Laboratory*, kurz *Tesla*, das im Zuge der Anfertigung dieser Arbeit vom Autor mitentwickelt wurde. Die Darstellung orientiert sich dabei vorwiegend an den Konzepten, die in Hinsicht auf die Funktionalität des Systems als Arbeitsplatz für den Textwissenschaftler umgesetzt wurden. Eine umfangreiche Beschreibung des Tesla-Systems aus technischer Sicht findet sich in der parallel entstehenden Arbeit des weiteren Systementwicklers Schwiebert (2011).

Kapitel 4 widmet sich der Beschreibung eines Textes, der im weiteren Verlauf der Arbeit mit Tesla prozessiert wird. Dieser Text befindet sich auf einem Anfang des 20. Jahrhunderts aufgefundenen Manuskript, dessen Inhalt sich bis zum heutigen Tag noch niemandem erschlossen hat. Es handelt sich hierbei um das weithin bekannte (und noch mehr berüchtigte) nach seinem Entdecker Winfrid Voynich benannte Voynich-Manuskript. Die Analyse des Manuskripttextes ist aus mehreren Gründen geeignet, die Möglichkeiten, die sich durch die Arbeit mit Tesla ergeben, zweckmäßig darzustellen. Zum einen wurden von einer weitverzweigten Community sehr unterschiedliche Analysen zu diesem Text durchgeführt, die innerhalb von Tesla nachvollzieh- und erweiterbar zusammengeführt werden können. Zum anderen lässt sich eine ganze Bandbreite von Methoden, die für sehr unterschiedliche wissenschaftliche Bereiche entwickelt wurden, zur experimentellen Prozessierung dieses unerschlossenen Textes einsetzen, wodurch das Potential von Tesla, vielfältige Verfahren zu integrieren, an einem konkreten Beispiel aufgezeigt werden kann.

Kapitel 5 bereitet den theoretischen Rahmen für die computationelle Verarbeitung des Voynich-Manuskripts. Dies geschieht durch eine Einführung in die Kryptologie, die Wissenschaft von der Beschäftigung mit Verschlüsselungsverfahren, sowie die Aufführung unterschiedlicher kryptographischer Methoden, deren Aufbau möglicherweise Rückschlüsse auf den Text des Voynich-Manuskripts erlaubt.

Kapitel 6 führt die Stränge der vorherigen vier Kapitel zusammen, indem das System Tesla angewendet wird, um Experimente hinsichtlich der Bestimmung des Verschlüsselungsverfahrens und der möglichen Schlüssel, die bei der Entstehung des Voynich-Manuskriptes zur Anwendung kamen, durchzuführen. Dabei werden unterschiedliche Verfahren, die innerhalb verschiedener textprozessierender Wissenschaften entwickelt wurden, auf den Text des Manuskriptes, den Text einer Vergleichschiffre und natürlichsprachliche Texte angewendet.

Kapitel 7 schließlich stellt noch einmal die zentralen in dieser Arbeit vertretenen Thesen heraus und gibt einen Ausblick auf die Anschlussfähigkeit der verwendeten Werkzeuge, Methoden und Verfahren.

In Jewish gnostic philosophy the universe consists of God's thought; thought is expressed in speech; speech is composed of letters; hence the Letters are the ultimate constituents of Things.

(R. G. Kent: The Cipher of Roger Bacon)

Kapitel 2

Textprozessierung

Im Jahr 1960 druckt das New Yorker *Time Magazine* einen Artikel, der sich mit den damals aktuellen Fortschritten hinsichtlich der Zuordnung von DNA-Basen zu Aminosäuren auseinandersetzt. Bemerkenswert an diesem Bericht ist weniger sein Inhalt als dass er unter dem Titel "Genetic Rosetta Stone"[1] erscheint, womit eindeutig auf die damals – sowohl in der Öffentlichkeit als auch in der Wissenschaft – sehr populäre Parallele zwischen Kryptoanalyse, Linguistik und Genetik Bezug genommen wird. Der Rosettastein wurde 1799 während einer napoleonischen Ägypten-Expedition von einem französischen Offizier im Niltal bei der Stadt Rashid (europäisch auch Rosette genannt) gefunden. Der Umstand, dass auf diesem Stein ein Text parallel in gleich drei verschiedenen Schriftsystemen (Altgriechisch, Demotisch, Hieroglyphen) eingemeißelt war, von denen zumindest eine bekannt war, trug maßgeblich zur Entzifferung der Hieroglyphen bei, die vorher jahrhundertelang nicht übersetzt werden konnten. Treibende Kraft der Entzifferung war der junge französische Archäologe Champollion, der die althergebrachte These verwarf, dass Hieroglyphen ideographisch zu analysieren seien und stattdessen annahm, es handle sich – zumindest zum Teil – um eine phonetische Schrift. Über die Ableitung bekannter Herrschernamen, die kryptoanalytische Methode der Häufigkeitsanalyse sowie Tests über geratene Kombinationen gelang es ihm schließlich 1822 ein komplettes System der Hieroglyphen aufzustellen, was noch immer als Meilenstein sowohl der Kryptoanalyse wie auch der Ägyptologie gefeiert wird.[2]

Es ist ein solcher Durchbruch, auf den die Biologie im Jahr 1960 noch wartet; zwar ist seit Watson & Crick (1953) die Struktur des Trägers der Erbinformation (der Desoxyribonu-

1 Time 23.5.1960, S. 50; siehe `http://www.time.com/time/magazine/article/0,9171,827622,00.html`, zuletzt aufgerufen am 11.10.2011.

2 Für Champollion war der Rosettastein lediglich eines von mehreren Puzzleteilen, die zu seinem System der Hieroglyphen beitrugen. Eine ausführliche Schilderung von Champollions Leben sowie der Umstände, die zur Entschlüsselung der Hieroglyphen beigetragen haben, findet sich in Adkins & Adkins (2000).

kleinsäure, kurz DNA) aufgeklärt,[3] unbekannt war allerdings weiterhin der Mechanismus, wie mit Hilfe dieser DNA *Proteine* (auch als *Bausteine des Lebens* bezeichnet) produziert werden können. Die Entdeckung, dass sich die DNA-Struktur auf eine doppelsträngige, lineare Abfolge von vier unterschiedlichen Basen zurückführen lässt, hatte zur Folge, dass von einem *DNA-Text* (oder gleich einer *DNA-Sprache des Lebens*), später auch von einem *DNA-Code* gesprochen wurde. Die von der Erbinformation gesteuerten Prozesse nachvollziehen zu können wäre demnach nichts anderes als den Code zu brechen, und dafür wäre ein *Stein von Rosette* für die *Sprache des Lebens* sehr hilfreich.

Mit der Analyse von Codes beschäftigt sich die Kryptologie, genauer die Kryptoanalyse. Deren Techniken spielten im zweiten Weltkrieg eine bedeutende (manche meinen mit kriegsentscheidene) Rolle und bildeten auch einen der Grundpfeiler für die Entwicklungen neuartiger Kommunikationstheorien und -technologien der Nachkriegszeit, namentlich Kybernetik (Wiener 1948), Informationstheorie (Shannon & Weaver 1949) und Computerwissenschaft (von Neumann 1945). Die vergleichsweise große Beachtung, die man in den 1950er und 1960er Jahren diesen Technologien entgegenbrachte, führte auch dazu, dass Forschungsfelder wie die Genetik und die strukturalistische Sprachwissenschaft als auf einer allgemeinen Kommunikations- und Informationstheorie aufbauende Wissenschaften uminterpretiert wurden. Auch die mit der Erfindung von Rechenmaschinen aufkommende Computerlinguistik, vor allem ihr Anwendungsfeld *Maschinelle Übersetzung* wurde als kryptoanalytische Disziplin angesehen; diese Auffassung wurde vor allem in dem sehr einflussreichen Weaver-Memorandum (Weaver 1949) vertreten, welches allgemein als Startschuss für die Maschinelle Übersetzung angesehen wird und dafür sorgte, dass der Bereich mit üppigen Finanzmitteln ausgestattet wurde (vgl. Hutchins 1986: Kap. 2.3ff). Eine der zentralen Aussagen des Weaver-Memorandums ist die folgende:

> "It is very tempting to say that a book written in Chinese is simply a book written in English which has coded into the 'Chinese code.' If we have useful methods for solving almost any cryptographic problem, may it not be that with proper interpretation we already have useful methods for translation?"

Weaver weist damit auf den entscheidenden Vorteil der Rückführung einzelner Wissenschaften auf eine gemeinsame Basis hin: Es wird möglich, Methoden, die innerhalb einer spezifischen Wissenschaft erfolgbringend entwickelt wurden, auf andere Wissenschaften zu übertragen und jene damit in unterschiedlichen Kontexten nutzbar zu machen. Auf

3 Die Geschichte der Entdeckung der Doppelhelix-Struktur der DNA zeichnet Watson (1980) auf sehr anschauliche Weise nach.

dem Höhepunkt dieser Anschauung wurde die gesamte Welt als Kommunikationssystem aufgefasst: Demnach haben sowohl Natur als auch Kultur eine gemeinsame Schnittstelle – die Sprache, die sie zugleich als Werkzeug sowie als Objekt der Betrachtung, d.h. sowohl im Sinne von *techne* als auch von *episteme*, nutzen.[4]

Wie sich in den Folgejahren zeigen sollte, konnte sich die Auffassung nicht durchsetzen, dass die erwähnten Forschungsrichtungen auf eine allgemeine Theorie der Kommunikation ausrichtbar sind und dass damit eine allen gemeinsame Methodik der Forschung zugrunde liegen könnte. Dies lag zum einen daran, dass die erhofften Fortschritte ausblieben: Im Falle der Maschinellen Übersetzung sorgte der sogenannte ALPAC-Report für Ernüchterung;[5] in der Genetik wurde die DNA-Protein-Übersetzung nicht von denjenigen gelöst, welche über kryptoanalytische Methoden arbeiteten,[6] sondern von denjenigen, welche den größten materiellen Einfallsreichtum an den Tag legten.[7] Hinzu kam, dass die Informationstheorie bei der Anwendung auf die Genetik und die Sprachwissenschaft unzulässig frei interpretiert wurde: Der Informationsbegriff ist nach Shannon explizit nicht semantischer Natur und keinesfalls ein Grundstoff oder, wie es Cherry (1963:214) ausführt:

> "Die Signale übermitteln nicht Information, wie etwa Güterzüge Kohle transportieren. Wir sollten besser sagen: Die Signale besitzen, vermöge ihrer Fähigkeit zur Auswahl, einen Informationsgehalt."

Wenn also von der DNA als Träger der Erbinformation gesprochen wird oder von natür-

4 Dies bedeutet, dass die Analyse von Sprache sowohl betrieben wird, um die den Texten inhärente *Information* als auch um die *Struktur* der den Texten zugrundeliegenden Sprache zu identifizieren. Verbalisiert wurden diese Überlegungen vor allem in einer vom französischen Fernsehen organisierten Diskussion mit dem Namen "Vivre et Parler" (im September 1967), an welcher Mediziner François Jacob, der Biologe Phillippe L'Héritier, der Linguist Roman Jacobson sowie der Anthropologe Claude Lévi-Strauss teilnahmen, vgl. Kay (2000:307).

5 Vgl. Pierce & et al. (1966). Kritisiert wurden darin vor allem die bis dahin erreichten Ergebnisse in Verhältnis zu den hochgesteckten Zielvorstellungen. Die von Weaver ausgedrückte Auffassung, bei der Maschinellen Übersetzung würde es sich um ein simples (De-)Codierungsproblem handeln, hatte keinen Bestand mehr. Der Report machte vielmehr darauf aufmerksam, dass die Funktionsweise natürlicher Sprachen noch viel zu wenig verstanden sei, als dass eine vollautomatische Übersetzung zum damaligen Zeitpunkt überhaupt anzugehen wäre. In der Folge kam die Entwicklung der Maschinellen Übersetzung – zumindest in den USA – fast zum Stillstand.

6 In der ersten Reihe steht hier der sogenannte RNA-Krawattenclub, dem Wissenschaftler der unterschiedlichsten Coleur angehörten, um sich des *Codierungsproblems* anzunehmen; neben Watson und Crick waren das u.a. die Physiker Richard Feynman und George Gamow (vgl. `http://www.ambion.com/main/tieclub/flash/print_form/story_2.html`, zuletzt aufgerufen am 11.10.2011.)

7 In diesem Fall waren es die Biochemiker Nirenberg & Matthaei (1961), welche eine experimentelle Anordnung fanden, mit der sich feststellen ließ, dass RNA-Schnipsel, die lediglich eine einzige der vier Basen (in mehrfacher Ausführung) enthielten, Proteine codierten, die nur aus einer der 20 Aminosäuren aufgebaut sind; vgl. Kay (2000:255).

lichsprachlichen Äußerungen, mit denen Informationen zwischen Individuen ausgetauscht werden, so ist dieser Informationsbegiff ein definitiv anderer als der in der Informationstheorie genutzte. Wie Kay (2000:413f) zeigt, ergaben sich viele der Analogien zwischen den verschiedenen Wissenschaften nicht aus der Ähnlichkeit des behandelten Gegenstands, sondern als Folge der Nutzung gleicher Metaphern oder des Transfers von Methoden, also gleichsam *post hoc*. All dies zusammengenommen führte dazu, dass das wissenschaftliche Paradigma, Kryptoanalyse, Sprachwissenschaft, Computerlinguistik und Genetik ließen sich auf eine gemeinsame Basis zurückführen, vom wissenschaftlichen Mainstream nicht weiter verfolgt wurde.[8]

Der Boom, den die Kommunikationstheorie bis dahin erlebte, war vorerst beendet, man setzte nicht weiter auf den verstärkten Transfer von Methoden, sondern entfernte sich eher wieder voneinander. Damit geriet zum Teil auch das Wissen darüber in den Hintergrund, dass sich nach wie vor eine gemeinsame Basis finden ließ, denn die Untersuchungsgegenstände lassen sich allesamt in einer textuellen Repräsentation fassen. In jüngerer Zeit kann man auch wieder Ansätze des Methodentransfers beobachten, so ist es z.B. im Bereich der Bioinformatik gelungen, effiziente Algorithmen zum Zeichenkettenvergleich zu entwickeln, die mittlerweile auch in sprachwissenschaftlichen Vorhaben verwendet werden (s.u., 2.1). Hier knüpft die vorliegende Arbeit an, in der es um die *Prozessierung* textueller Daten geht. Ein *Prozess* ist allgemein nach DIN 19226 Teil 1 definiert als "Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder auch Information umgeformt, transportiert oder auch gespeichert wird." Spezifiziert auf textuelle Daten steht der Begriff *Textprozessierung* in dieser Arbeit für den Prozess der maschinellen Verarbeitung von Texten zum Zwecke des Informations- bzw. Wissensgewinns.[9] Diese Arbeitsdefinition lässt *Textprozessierung* zumindest partiell bedeutungsgleich zu den geläufigeren Begriffen *Textverarbeitung* und *Text Mining* erscheinen. Da aber das Kompositum *Textverarbeitung* gemeinhin als Werkzeug zur Erstellung von Texten angesehen wird,[10] während *Text Mining* meist mit einem spezifischen Teilbe-

8 Im Sinne der Wissenschaftstheorie von Kuhn (1970) könnte man davon sprechen, dass diese Anschauung nicht über die vorparadigmatische Phase hinausgekommen ist.

9 *Information* wird hier wie auch in der weiteren Arbeit nicht im Sinne der Informationstheorie, sondern im Sinne von Schmitz & Zucker (2003) für strukturierte Daten verwendet. Stark vereinfacht kann dann die Gesamtheit aller verfügbaren Informationen und ihrer wechselseitigen Zusammenhänge als *Wissen* angesehen werden.

10 Gemeint sind hier Editoren wie *Microsoft Word* oder *OpenOffice Writer*. Ferner werden in der Textlinguistik die komplementären Prozesse Textgenerierung und Textverarbeitung unterschieden. Textverarbeitung bezieht sich dabei auf die kognitiven Vorgänge Aufnahme, Transformation, Organisation, Spericherung, Reaktivierung und Reproduktion der Textinformation (Christmann 2000).

reich der Textprozessierung assoziert wird,[11] erscheint hier die Verwendung eines neuen Begriffs notwendig.[12]

Dieses einführende Kapitel wird zunächst aufzeigen, welche Wissenschaften überhaupt textuelle Daten verarbeiten, d.h. welche Disziplinen an Informations- oder Wissensgewinnung aus Texten interessiert sind (2.1). Auf Grundlage dieser Anwendungsbereiche wird der Begriff Text – so wie er in dieser Arbeit verwendet wird – definiert. In diesem Zuge werden auch die wichtigsten Eigenschaften von textuellen Daten aufgeführt (2.2). Damit werden die zuvor genannten Disziplinen auf eine gemeinsame Basis – die zugrundeliegende Datenstruktur Text – gestellt. Auf dieser Grundlage wird in 2.3 eine offene/generalisierte textprozessierende Wissenschaft fundiert. Die Frage, welche Arbeitsumgebung eine solche Wissenschaft idealerweise benötigt, behandelt der finale Abschnitt dieser Einführung 2.4 und leitet damit auf das anschließende Kapitel 3, zur Instanz einer solchen Arbeitsumgebung, dem *Text Engineering Software Laboratory*, über.

2.1 Forschungsfelder der Textprozessierung

Spricht man von Texten, so wird darunter in der Regel verschriftete natürliche Sprache verstanden. Traditionell werden deshalb vor allem Sprach- und Literaturwissenschaft als textverarbeitende Disziplinen angesehen. Tatsächlich, wie eingangs gezeigt, können auch nicht-natürlichsprachliche Daten durch eine lineare Abfolge von Symbolen repräsentiert werden. Die Art der Information, die in den Daten steckt, weicht in den verschiedenen Wissenschaften voneinander ab. Welche Wissenschaften mit Texten arbeiten und welche Informationen sie aus diesen erschließen wollen, wird im folgenden kurz dargestellt.

11 Feldman & Sanger (2006:1) definieren den Begriff Text Mining in Hinsicht auf natürlichsprachliche Daten: "Text mining can be broadly defined as a knowledge-intensive process in which a user interacts with a document collection over time by using a suite of analysis tools." Heyer *et al.* (2006:5) unterscheiden in einer Kreuzklassifikation unstrukturierte und strukturierte Daten, die jeweils abgefragt oder angefragt werden können. Texte zählen nach diesem Schema zu unstrukturierten Daten (vgl. dazu die Verwendung des Begriffs Semistruktur in 2.2). Die inhaltliche *Abfrage* der Informationen in unstrukturierten Daten wird durch Suchmaschinen und Dokumentenmanagementsysteme geleistet, die inhaltliche *Anfrage* durch Text Mining Werkzeuge. Textprozessierung ist insofern als Oberbegriff für Text Mining anzusehen.

12 Textprozessierung ist auch verwandt mit dem in jüngerer Zeit aufgekommenen Ausdruck *Texttechnologie*. Texttechnologische Verfahren sind nach Lobin & Lemnitzer (2003) in vier Arten zu klassifizieren: Informationsanreicherung (Annotation), Informationsextraktion (Strukturierung der Information), Information Retrieval/Textmining (Auffinden von Information) und Erstellung neuer, vom Primärtext abgeleiteter Objekte. Alle diese Bereiche sind auch Instanzen der Textprozessierung, welche damit als Oberbegriff für texttechnologische *Verfahren* angesehen werden kann.

Gegenstand der **Sprachwissenschaft** sind, wie der Name schon andeutet, natürliche Sprachen. Der Begriff Sprache ist unscharf, er wird auf recht unterschiedliche Weise gebraucht. Ferdinand de Saussure, Begründer der modernen theoretischen Sprachwissenschaft, unterschied zwischen der allgemeinen Fähigkeit, Sprache anzuwenden (*faculté de language*), der Beherrschung eines bestimmten Sprachsystems (*langue*) und der Verwendung dieses Systems (*parole*). Zuvor galt jahrhundertelang die Analyse schriftlich fixierter Texte als das vorrangige Betätigungsfeld für – in dieser Zeit noch der präskriptiven Ausrichtung verpflichteten – Sprachwissenschaftler. Dies änderte sich mit de Saussure, der Schriftsprache lediglich als System ansah, das nur zu dem Zweck besteht, ein anderes System, nämlich das der gesprochenen Sprache, darzustellen (de Saussure 1916:28). Fortan galt das *Primat der gesprochenen Sprache*, da diese in der menschlichen (onto- und phylogenetischen) Entwicklung vor der Schriftsprache erworben wird; viele Sprachgemeinschaften haben gar nie eine Verschriftlichung ausgebildet. Im Zuge der *kommunikativ - pragmatischen Wende* der Linguistik, sich die unter der Einwirkung pragmatischer und soziolinguistischer Theorien herausbildete, wurden zu Beginn der 1970er Jahre die Besonderheiten der gesprochenen Sprache gegenüber der Schriftsprache herausgestellt (Watzlawick *et al.* 2007). So ist man sich in der Linguistik mittlerweile wieder einig darüber, dass es sich bei gesprochener Sprache und Schriftsprache nicht nur um unterschiedlichen Sprachgebrauch, sondern tatsächlich um unterschiedliche Sprachsysteme handelt. Mit der Schriftsprache ist es möglich geworden, Gedanken in Texten zu konservieren und auf diese Weise weite räumliche und zeitliche Entfernungen zu überwinden. Sprachliche Äußerungen lassen sich damit vom ortsgebundenen Augenblick lösen und sind folglich immer wieder konsumierbar. Beruhend auf diesem und weiteren (im Verlauf dieses Kapitels aufzuzeigenden) Vorzügen fanden schriftsprachliche Zeugnisse Eingang in den empirisch orientierten Zweig der Sprachwissenschaft, der auch als *Korpuslinguistik* bezeichnet wird.[13] Im Bereich der Korpuslinguistik werden Theorien anhand tatsächlich vorhandener Sprachdaten (Texte oder – vorzugsweise gewählt, aber meist nicht im entsprechenden Umfang verfügbar – transkribierte verbale Äußerungen) überprüft. Im Gegensatz dazu steht die kognitive Schule, entstanden durch die kognitive Wende der Sprachwissenschaft mit Chomsky (1957): Für diese sind Sprachdaten (bei Saussure *parole*, bei Chomsky *Perfor-*

13 McEnery, Wilson (1996) weisen darauf hin, dass das Kompositum *Korpuslinguistik* kein Teilbereich der Linguistik, wie Syntax oder Semantik, aber auch keine der typischen "Bindestrich"-Linguistiken ist, die nur auf einen bestimmten Anwendungsbereich (Psycho-, Sozio-, Rechtslinguistik etc.) ausgerichtet sind. Stattdessen ist Korpuslinguistik *Methode*, und zwar die den anderen Methoden vorzuziehende (s.u.).

manz), teilweise defektive Daten, die keine direkten Rückschlüsse auf das Sprachssystem (bei Saussure *langue*, bei Chomsky *Kompetenz*[14]) zulassen. Kompetenz ist die Fähigkeit eines Menschen (Muttersprachlers), eine Sprache zu beherrschen. Chomsky sieht sie gleichsam als Organ, das sich aus einer genetisch determinierten Vorstufe (der *Universalgrammatik*) in Konfrontation mit einzelsprachlichen Performanzdaten zu einer Fähigkeit, eine bestimmte Sprache zu beherrschen, ausbildet. Entscheidend für die Rückschlüsse auf das Sprachsystem sind damit keine real existierenden Sprachdaten. Stattdessen werden Hypothesen über Eigenschaften des Sprachsystems über Introspektion gewonnen, d.h. der idealisierte Sprecher/Hörer (eigentlich: der Sprachwissenschaftler) entscheidet, ob es sich um korrekte oder inkorrekte Daten handelt.[15] Es kann und soll hier keine abschließende Wertung darüber gegeben werden, welche der widerstreitenden Schulen der Sprachwissenschaft mit der richtig(er)en Methode arbeitet. So plausibel die Annahmen der generativen Schule auch sind, ist es ihr auch nach über einem halben Jahrhundert nicht gelungen, ein Modell zu entwickeln, welches den von Chomsky selbst formulierten Ansprüchen an Universalität und Lernbarkeit genügt.[16] Auf den ersten Blick scheint sich der in dieser Arbeit vertretene Ansatz eindeutig auf der empirischen Seite zu positionieren. Seit einiger Zeit gibt es aber auch Bestrebungen, die Errungenschaften beider Schulen – der empirischen

14 Das Sprachsystem ist bei Saussure ein statisches, während Chomsky die Kompetenz auf ein dynamisches System bezieht, das aus einer Menge von Erzeugungsprozeduren besteht. Deshalb sind die beiden Begriffe nicht als synonym zu betrachen.

15 Fillmore (1992:35) karikiert einen solchen Wissenschaftler auf sehr prägnante Weise: “He sits in a deep soft comfortable armchair, with eyes closed and his hands clasped behind his head. Once in a while, he opens his eyes, sits up abruptly shouting, ’Wow, what a new fact!’, grabs his pencil, and writes something down. Then he paces arround for a few hours in the excitement of having come still closer to knowing what language is really like.” Eine ähnlich wertende Passage findet sich in McEnery & Wilson (1996:1) (die Autoren distanzieren sich insofern von der Aussage, als dass sie sie sicherheitshalber in den Bereich der Folklore stellen): “Corpus Linguists study real language, other linguists just sit at their coffee table and think of wild and impossible sentences.”

16 Innerhalb der generativen Theorie existiert eine nicht leicht zu durchschauende Vielzahl unterschiedlicher Schulen. Im Abstand von 10-15 Jahren veröffentlicht Chomsky selbst weiterentwickelte Modelle, die mit den vorhergehenden über weite Strecken inkompatibel sind. Das ursprüngliche Modell aus den “Syntactic Structures” (Chomsky 1957) wurde vom *Standardmodell* abgelöst (Chomsky 1965), worauf das *Prinzipien- und Parameter-Modell* folgte, das zuerst innerhalb der *Rektions- und Bindungstheorie* (englisch Government & Binding, kurz GB, Chomsky 1981), später innerhalb des *Minimalistischen Programms* (MP, Chomsky 1995) interpretiert wurde. Chomsky verfolgt vor allem das Ziel, strukturelle (syntaktische) Eigenschaften konsistent (und damit maschinell anwendbar) mit inhaltlichen (semantischen) Eigenschaften in Bezug zu setzen. Dass dies nicht vollständig gelungen ist, lässt sich auch daran ablesen, dass GB und MP in der maschinellen Sprachverarbeitung nur sehr sporadisch eingesetzt werden. Ausnahmen bilden die von Rolshoven (1987) entworfene, an der GB-Theorie orientierte *Linguistische Programmiersprache* (LPS), die im gleichnamigen Maschinellen Übersetzungssystem (Rolshoven 1991) und der Grammatik-Modellierungsplattform *VisualGBX* (Lalande 1997) eingesetzt wird, sowie die PROLOG-Implementierung eines Teils der GB-Theorie von Dougherty (1994).

und der kognitiven – miteinander in Einklang zu bringen, bzw. die Ergebnisse wechselseitig zu nutzen (angefangen mit diversen Ansätzen in Klavans & Resnik 1996). Im weiteren Verlauf werden wir darauf zurückkommen, dass sich gerade für einen solchen übergreifenden Ansatz eine Infrastruktur wie die in dieser Arbeit vorgestellte eignet.

Die maschinelle Verarbeitung linguistischer Daten ist durch die Sprachwissenschaft motiviert, dem allgemeinen Verständnis nach aber dem Bereich der **Computerlinguistik** (CL) zuzuordnen. Neben der Realisierung linguistischer Methoden auf Rechenmaschinen, sei es zur Überprüfung linguistischer Theorien oder zur Aufdeckung von sprachlichen Strukturen, schließt die CL auch die Entwicklung praxisorientierter Sprachsoftware ein. Dieser Teilbereich wird auch als **Sprachtechnologie** bezeichent. Dabei rücken hauptsächlich die Anwendungsbereiche der *Informationsgewinnung* in den Fokus wissenschaftlicher und insbesondere wirtschaftlicher Interessen. Grund dafür ist, dass die gewaltige Menge an Information, die inzwischen im WorldWideWeb (WWW)[17] vorliegt, in sprachlicher Form und zum ganz überwiegenden Teil in natürlichsprachlichen Texten codiert ist. Um diese Informationen maschinell erschließen zu können, bedarf es auch des Wissens über die sprachliche Codierung. Dies hat dazu geführt, dass die Computerlinguistik sich verstärkt auf Prozesse der Gewinnung von Information aus un- bzw. semistrukturierten (s.u. Kapitel 2.2) Texten in den Anwendungsbereichen *Information Retrieval*, *Information Extraction*, *Text Classification*, *Text Summarizing* und *Text Mining* ausgerichtet hat. Diese Techniken werden praktisch in sämtlichen wissenschaftlichen Bereichen genutzt, um der immer größeren Zahl von – textlich codierten – wissenschaftlichen Publikationen Herr zu werden.[18] Vor allem die sehr dynamischen Bereiche der medizinischen und biowissenschaftlichen Forschung sind auf Methoden, die Flut an neuen Forschungsergebnissen zu ordnen und zugänglich zu machen, angewiesen.[19]

17 Dies umschließt nicht nur das über Suchmaschinen öffentlich zugängliche *Surface Web*, sondern auch das sogenannte *Deep Web*, das zum großen Teil aus Fachdatenbanken und dynamisch (aus Suchanfragen an Datenbanken) generierten Webseiten besteht (vgl. Bergman 2001). Schätzungen zufolge umfasst das Deep Web eine weitaus größere Datenmenge als das Surface Web.

18 Neben dieser Informationsstrukturierung wird mitunter auch in nicht primär sprachwissenschaftlich ausgerichteten Disziplinen die Struktur von Sprache untersucht, wie etwa in der Politikwissenschaft, wo neuerdings – mit vergleichsweise einfachen Mitteln wie der Frequenzanalyse von Diskussionsbeiträgen – bei Debatten für einzelne Politiker oft wiederholte Terme ermittelt (`wortwahl.zdf.de`, aus rechtlichen Gründen nicht mehr verfügbar) oder politische Positionen über die Analyse von Texten gewonnen werden (Laver & Benoit 2003). Auch die Literaturwissenschaft setzt die erwähnten Verfahren zur maschinell unterstützten Analyse literarischer Texte ein, z.B. im Projekt *WordSeer* der Universität Berkeley `www.cs.berkeley.edu/~aditi/projects/wordseer.html`, zuletzt aufgerufen am 11.10.2011.

19 Exemplarisch sei hier das *BioCreAtive*-Projekt genannt, in dem Information Retrieval, Informa-

Es wurde bereits erwähnt, dass die CL initial auf Anstrengungen zur *Maschinellen Übersetzung* zurückzuführen ist, die ihrerseits ihren Ausgangspunkt in den großen Forschritten hatte, welche im Bereich der **Kryptologie** während und direkt nach dem zweiten Weltkrieg erzielt wurden. Kryptologie beschäftigt sich allgemein mit technischen Verfahren der Informationssicherheit, ihr bekanntester Anwendungsbereich ist die Codierung und Decodierung von textueller Information. Kryptographische Methoden können den Personenkreis einschränken, für den die Information zugänglich ist. Vor allem für den unautorisierten Zugriff auf solcherlei verborgene Information über kryptoanalytische Methoden spielte in früherer Zeit (d.h. vor dem Aufkommen von Rechenmaschinen und der Entdeckung mathematischer Funktionen, die nahezu sichere Verschlüsselungen ermöglichen) die quantitative Analyse verschlüsselter Texten eine große Rolle. Durch die spezielle Anatomie natürlicher Sprachen (Auftritt spezifischer Muster und Häufigkeitsverteilungen) ergeben sich beachtliche Einfallstore für den Angriff auf unzureichend gesicherte Texte.

Textprozessierende Wissenschaften erschöpfen sich nicht notwendigerweise in solchen, die natürlichsprachliche Texte als Gegenstand haben. Letztlich kann man jedwede Wissenschaft, deren zu analysierende Daten sich linearisieren und in einzelne Einheiten zerlegen lassen, als textprozessierende Wissenschaft konstruieren. So weist u.a. Raible (2001) darauf hin, dass auch in der Biologie, speziell in der **Genetik**, der Begriff Text nicht nur als Metapher gebraucht wird, wenn vom genetischen Code gesprochen wird, sondern es – bei allen Unterschieden – auch eine Reihe von Analogien zwischen natürlichsprachlichen und genetischen Texten gibt. Beide Systeme – der biologisch-genetische Code wie auch die menschlichen Sprachsysteme – stehen vor der gleichen großen Herausforderung, eine im Grunde komplex-vieldimensionale Struktur (Aufbau von Organismen bzw. Schilderungen von Situationen) in einer eindimensional-linearen Form codieren zu müssen. Auf Basis dieser Gemeinsamkeit erwächst die Hoffnung, dass Algorithmen die zur Aufdeckung von Mustern innerhalb einer textprozessierenden Wissenschaft entwickelt wurden, auch in anderen Bereichen nutzbar sind. Zu derartigen Übertragungen wurden und werden bereits viele verschiedenartige Ansätze verfolgt, z.B.:

- Die Forschergruppe *Machine Learning and Applied Statistics* von Microsoft Research nutzte den für die Erkennung von Spam-Emails entwickelten Teiresias-Algorithmus um potentielle Epitopen[20] in Proteinen aufspüren können. Die „Mutationen“ auf Zei-

tion Extraction und Text Mining auf biologische Publikationen angewendet werden, siehe `http://biocreative.sourceforge.net/`, zuletzt aufgerufen am 11.10.2011.

20 Epitopen sind Bereiche von Proteinen, gegen die das Immunsystem Antigene bildet.

chenebene, die ein Spamfilter beachten muss, halfen bei der Vorhersage tatsächlicher Mutationen des HI-Virus (Jojic *et al.* 2005).

- Kay (2004) zeigt, dass Methoden, die lange Zeit vor allem in der Domäne der Bioinformatik weiterentwickelt wurden, wie *Suffix-Bäume* und *Alignment-Verfahren*, auf natürlichsprachliche Daten angewendet werden können.
- Simon Shepherd, Wissenschaftler an der Universität von Bradford, will 2001 einen Algorithmus entwickelt haben, der es ermöglicht, aus natürlichsprachlichen Texten entfernte Spatien wieder zu rekonstruieren. Angewendet auf die DNA des Ebola-Virus soll ein verwandter Algorithmus Informationen über aktive Regionen der DNA liefern. "We are treating DNA as we used to treat problems in intelligence" lässt sich Shepherd in Nature zitieren (Pearson 2006).[21]
- Im Rahmen des interdisziplinären Forschungsverbundes Linguistik–Bioinformatik an der Humbold-Universität Berlin wurden phylogenetische Algorithmen auf die diachrone Sprach- und Dialektforschung angewendet (Hochmuth *et al.* 2008). Parallelen zwischen genetischer und sprachlicher Verwandschaft hat auch der Populationsgenetiker Cavalli-Sforza (1997) herausgearbeitet.
- Die Arbeitsgruppe *Plant Computational Biology* am Max-Plank-Institut für Pflanzenzüchtung (Köln)[22] nutzt Technologien aus dem Bereich des *Sematic Web*, um biologisches Wissen in elektronisch verarbeitbarer Form zu beschreiben mit dem Ziel, über die Verarbeitung großer Datenmengen Analogien zwischen Ergebnissen aufzudecken.

Neben natürlichsprachlichen und genetischen Texten finden sich auch solche, in denen Anweisungen für Rechenmaschinen in sogenannten Programmcodes persistiert sind. Solche Programmcodes sind entweder für Maschinen direkt interpretierbar, dann spricht man von *Maschinensprache*, oder sie sind in einem *Quellcode* verfasst, der lesbar für Menschen ist, jedenfalls für diejenigen, die der Programmiersprache mächtig sind, in der dieser Quellcode formuliert ist. Eine Zwischenstufe ist der sogenannte *Bytecode*, der von einigen interpretierten Programmiersprachen aus dem Quellcode erzeugt wird, um Programme in eine kompakte Form zu bringen, die unabhängig von der Maschine ist, auf der der Code schließlich ausgeführt wird. Eine solcher Bytecode ist damit portabel. Un-

21 Obwohl diverse (auch wissenschaftliche) Medien über diesen Algorithmus oft berichteten, gibt es dazu anscheinend keinen zitierfähigen Aufsatz von Shephard selbst. Auch lässt sich der Algorithmus nirgendwo einsehen.

22 `http://www.mpiz-koeln.mpg.de/forschung/unabhaengigeForschergruppen/Schoof/index.html`, zuletzt aufgerufen am 11.10.2011.

abhängig von der Kompilierung, der Interpretation und letztlich der Ausführung dieser Texte auf Computern (die auf textprozessierenden Aufgaben beruhen) werden im Bereich der **Softwaretechnologie** auch text-interpretierende Verfahren eingesetzt, in modernen Entwicklungsumgebungen (IDEs) etwa die automatische Formatierung des Quelltextes, die partielle Kompilierung, um eventuelle Programmierfehler umgehend zu bemängeln oder vorher festgelegte Programmier-Konventionen über *Style Checker* einzufordern. Eine weitere Anwendung textprozessierender Verfahren über Quelltexten ist die *statische Codeanalyse*, über die inhaltliche Fehler des Programmcodes (solche, die nicht durch den Compiler erkannt werden) werkzeuggestützt oder vollautomatisch erkannt oder zumindest eingegrenzt werden können. Jenseits der Analyse von Programm-Texten werden Analyseverfahren auch auf computer-generierte Texte angewendet, etwa bei der Auswertung von Logging-Dateien z.B. im Bereich der Web-Analytics.

Wie aus der Vielfalt dieser kurzen und keinesfalls erschöpfenden Aufzählung[23] von textprozessierenden Wissenschaften ersichtlich ist, kann nur ein Textbegriff, der sehr weit definiert wird, allen diesen Disziplinen gerecht werden.

2.2 Texte als Untersuchungsgegenstand

Bestrebungen, den Begriff Text zu definieren, wurden vorwiegend in der Textlinguistik bzw. der Texttheorie unternommen. Der klassische Strukturalismus, bei dem alle sprachlichen Phänomene letztlich auf die Basis der grundlegenden Operationen Segmentierung und Klassifizierung zurückgeführt wurden, kannte nur die linguistischen Einheiten Phonem, Morphem, Wort, Wortgruppe (Phrase) und Satz. Damit endete die linguistische Beschäftigung jenseits der Satzgrenze (vgl. Bloomfield 1935:170). Dies gilt in weiten Teilen auch noch für die nach der kognitiven Wende den Strukturalismus als herrschendes Forschungsparadigma beerbende Generative Grammatik. Dabei gab es schon im Distributionalismus[24] Bestrebungen, die Struktur von Texten aufzudecken (etwa bei Harris 1952),

23 So werden z.B. auch im Bereich der **Musikwissenschaft** Verfahren angewendet, die auf Texten (Partituren) operieren (z.B. *Music Information Retrieval*).

24 Als Distributionalismus wird die Phase des amerikanischen Strukturalismus Mitte des 20. Jahrhunderts bezeichnet, in der die Analyse von Sprachen über eine Distributionsanalyse betrieben wurde. Sprachliche Einheiten werden dabei aus der Untersuchung ihrer Kombinierbarkeit und gegenseitiger Ersetzbarkeit gewonnen, siehe auch Kapitel 6.2. Nicht zuletzt ist es mit neueren computergestützten Methoden möglich geworden, derartige, bereits in den 1950er Jahren skizzierte strukturalistische Aufdeckungsverfahren umzusetzen, wie dies etwa im *Alignment Based Learning* (ABL, van Zaanen 2000) geschieht. Des Weiteren wurden in vergleichsweise jüngerer Zeit Verfahren für die numerische Repräsentation textueller Daten entwickelt, z.B. das *Vector Space Model* (VSM, entwickelt von Gerald

auch um satzübergreifende Referenzstrukturen adäquat behandeln zu können (vgl. Koch 1966).

Später rückte in den Vordergrund, Kriterien auszumachen, welche Texte als solche zu definieren vermögen. Tatsächlich aber gelang es bisher nicht, eine allgemein anerkannte Definition von Texten zu finden. Grundlage für Texte im linguistischen Sinn ist der Begriff der Textualität, für den meist die sieben kommunikationsbezogenen Kriterien von de Beaugrande & Dressler (1981) herangezogen werden. Als die beiden zentralen Kriterien gelten dabei *Kohäsion* und *Kohärenz*. Kohäsion bezeichnet die grammatischen Abhängigkeiten, die für die syntaktische Verbindung der Elemente eines Textes verantwortlich sind, Kohärenz die semantisch-kognitiven Konzepte, bspw. Kausalitäts- und Referenzbeziehungen, welche für die Sinnkontinuität eines Textes sorgen. Die restlichen von den Autoren angeführten Textualitätskriterien sind Intentionalität, Akzeptabilität, Informativität, Situationalität und Intertextualität. Vater (2001:46ff) zeigt, dass die meisten dieser Kriterien für einen linguistischen Textbegriff problematisch sind und dass keines als notwendig oder gar hinreichend für eine Textbewertung herangezogen werden kann.

Aus Sicht der Informatik sind Texte *semistrukturierte* Daten, da sie einen Teil der Information nur implizit tragen (bspw. Gliederung durch unterschiedliche Formatierungen). Bestrebungen, diese implizite Information explizit zu erfassen, resultierten in der Einführung von Dokumentengraphen, die sich besonders einfach in unterschiedlichen Markup-Sprachen repräsentieren lassen (Morawietz & Mönnich 2004). Zudem wurde der Textbegriff, der traditionell eher als unveränderliche kommunikative Okkurenz interpretiert wurde, durch das Aufkommen neuer potentiell interaktiver Medien wie dem Internet hin zum *Hypertext*[26] erweitert. Hypertexte zeichnen sich dadurch aus, dass Textteile durch Hyperlinks miteinander verknüpft werden können und so die ursprünglich vom Papiermedium geforderte lineare Textstruktur aufbrechen. Hypertexte entsprechen Graphen, in denen man fast beliebig zwischen den einzelnen Textabschnitten navigieren kann, weil diese modularisiert und selektiv abrufbar sind. Hypertexte sind zudem immer compu-

Salton in den 1970er Jahren, ohne es in einer einschlägigen Veröffentlichung niederzulegen, vgl. Dubin 2004), der *Hyperspace Analoge to Language* (HAL, Lund & Burgess 1996) sowie die Hauptkomponentenanalyse (verwendet im *Latent Semantic Indexing* bzw. bei der *Latent Semantic Analysis*, LSI, Deerwester *et al.* 1990). Diese Methoden können genutzt werden, um über Klassifikationsverfahren, wie *Naive-Bayes Klassifikatoren* oder *Support Vector Machines*[25] Strukturen des den Daten zugrundeliegenden Sprachsystems zu ermitteln.

26 Der Begriff geht zurück auf einen Vortrag des Harvard-Soziologen/Philosophen Ted Nelson von 1972, als geistiger Vater der Hypertextidee gilt aber Vannevar Bush, der schon 1945 ein System erdachte, in dem Dokumenten(teile) miteinander verknüpft werden sollen (vgl. Storrer 2004:25.)

terbasiert, da das alternative Medium Papier die interaktiven und individualisierbaren Komponenten von Hypertexten nicht abbilden kann.[27]

Für die Definition eines möglichst generischen Textbegriffs, der alle Untersuchungsobjekte der in 2.1 unvollständig aufgezählten textprozessierenden Einzelwissenschaften zu fassen vermag, reicht der oben erläuterte linguistische Gegenstand nicht aus. Er mag für weite Teile der Literaturwissenschaft eine Rolle spielen, aber schon in der Korpuslinguistik, wo Korpora bisweilen aus einzelnen, unzusammenhängenden Sätzen bestehen, kann von über Satzgrenzen hinausgehender Kohäsion oder Kohärenz nicht mehr die Rede sein. Es sollte auch unbestritten sein, dass sich dieser Textbegriff schwerlich auf das Genom oder Partituren anwenden lässt (wenngleich es gewiss interessant wäre, dies zu untersuchen). Benötigt wird also eine Definition von Texten im weiteren Sinn. Betrachten wir die Gemeinsamkeiten der Texte aus 2.1: Sämtliche erwähnten Textsorten sind stets in diskreten (also nicht stetigen) Symbolen codiert[28] und können zu einer linearen Folge kombiniert werden. Im Gegensatz zu den Begriffen Zeichenkette oder String (aus dem Bereich der Programmiersprachen) referiert der Ausdruck Text nicht auf einen abstrakten Datentyp, sondern auf tatsächlich existierende Objekte der Welt. Aus diesen Überlegungen ergibt sich folgende weitgefasste Textdefinition, welche die Grundlage für die weiteren Ausführungen bildet:

Definition 1 (Text) *Ein Text ist eine konkrete Instanz einer Sequenz diskreter Einheiten aus einem endlichen Alphabet.*

Texte sind eine Form der Repräsentation, bei der in den meisten Fällen ein Teil der ursprünglich im repräsentierten Objekt vorhandenen Information verloren geht. Wie oben schon angeführt wurde, wird in der Linguistik von altersher die gesprochene Sprache als primäres Untersuchungsobjekt, die geschriebene als sekundär oder nachgeordnet betrachtet. Bei der Verschriftlichung gehen die Informationen über die Satzmelodie, die Betonung oder die mit dem Sprachsignal korrespondierende Mimik bzw. Gestik verloren.[29] Bisweilen

27 Genauer ausgedückt müsste es heißen: Papier kann Hypertexte nicht besonders gut abbilden. Natürlich kann man auch innerhalb von gedruckten Werken Querverweise einbringen. Die Referenz dieser Verweise erreicht man aber nicht komfortabel mit einem Mausklick, sondern durch eher mühsame manuelle Suche. Schon vor Verbreitung von Computern wurden z.B. nicht-lineare Romane veröffentlicht, wie etwa Georges Perecs *La Vie mode d'emploi*, deutsch *Das Leben – eine Gebrauchsanweisung.*

28 Hinsichtlich der Diskretheit typographischer Zeichen muss der Status von Ligaturen betrachtet werden. Diese sind teilweise als eigenständige Zeichen in das Grapheminventar aufgenommen worden (wie im Deutschen ß, w und &), teilweise werden sie aus ästhetischen Gründen für den Schriftsatz genutzt (in LaTeX etwa fl vs. fl). In fast allen indischen Schriften haben Ligaturen einen graphematischen Status, ihr Gebrauch ist also bedeutungsunterscheidend.

29 Tillmann (1997) weist allerdings auch darauf hin, dass transkribierte Texte Informationen enthalten, die sich nicht explizit aus dem Sprachsignal ablesen lassen (etwa Wortzwischenräume und Satzzeichen).

gibt es Ansätze, diese Information über Annotationen der Verschriftlichung wieder zuzuordnen, dies ist aber mit hohem Aufwand verbunden und es existiert dafür – im Gegensatz zum phonetischen Zeicheninventar IPA[30] – kein standardisiertes Format.[31] Neben dem Informationsverlust werden durch die Verschriflichung auch evtl. nicht zutreffende Abstraktionen getroffen, da nicht alle beliebigen Formate in diskrete Zeichen eines endlichen Alphabets zerlegt und dann linearisiert werden können. Sprachdaten sind aber zumindest zum Zeitpunkt ihrer Übertragung definitiv linear organisiert, d.h. sequenzierbar.

Anders verhält es sich bei genomischen Daten: Zwar liegt die Erbinfomation von Lebewesen (die DNA) als eine Abfolge komplementärer Basenpaare entlang eines Zucker-Phosphat-Bandes vor, die man scheinbar verlustlos in eine sequenzielle Abfolge diskreter Symbole (A für Adenin, C für Cytosin, G für Guanin und T für Thymin) übertragen kann. Bei dieser Übertragung gehen allerdings sämliche weiteren räumlichen Organisationsformen der DNA verloren: Die über die komplementären Basen verbundenen beiden Zucker-Phosphor-Bänder bilden zunächst eine Doppel-Helix; durch sequenzinduzierte Beugung (dabei bilden eine Reihe von aufeinanderfolgenden AT-Paaren einen Keil) kann diese Helix eine gekrümmte Struktur annehmen, die zu höheren Organisationsformen, wie etwa Nucleosomen, Chromatin und letztlich Chromosomen führt. Diese Auffaltung der Sequenz ist nicht nebensächlich, sondern führt zu physikalischen Angriffspunkten, an welche die zelluläre Maschinerie im Zuge der Genexpression andocken kann. Die räumliche Struktur wird zwar letztlich durch die Abfolge der Basenpaare konstituiert, ist durch reine (eindimensionale) Textanalyse heutzutage aber noch nicht exakt nachbildbar. Es ist ohnehin die Frage, ob die Erbinformation allein durch den DNA-Text codiert wird oder ob nicht die gesamte Maschinerie (d.h. die gesamte Zelle) für die Weitergabe verantwortlich ist.[32]

Aus den obigen Ausführungen folgt, dass die Codierung von Information in Sequenzen diskreter Einheiten, also Texten, diverse Nachteile hat: So kann in den seltensten Fällen die gesamte Information bewahrt werden, weil unzutreffende oder unerlaubte Abstraktionen getroffen werden (wie bei der Transkription gesprochener Sprache), oder von mehreren Dimensionen nur eine einzige übrig bleibt (wie bei der textlichen Repräsentation der DNA).

30 Kurz für *Internationales Phonetisches Alphabet* von der *International Phonetic Association*, die bereits 1886 in Paris gegründet wurde, vgl. `http://www.langsci.ucl.ac.uk/ipa/`, zuletzt aufgerufen am 11.10.2011.

31 Innerhalb der *Text Encoding Initiative* (TEI) wurde aber zumindest ein Rahmenwerk für die textuelle Repräsentation gesprochener Sprache entwickelt (vgl. Sperberg-McQueen & Burnard 2002: Kapitel 8).

32 Prominent wird die letztere Anschauung von Oyama (1985) vertreten, die darauf ihre Entwicklungssystemtheorie (Development Systems Theory) begründete.

Die Verwendung von Texten hat dessen ungeachtet aber auch eine Reihe von Vorteilen zu bieten: Texte bestehen aus diskreten Einheiten, sind stetig (im Sinne von nicht-flüchtig) und eindimensional, daher effizient speicherbar, eindeutig referenzierbar, indizierbar und mithin unkompliziert durchsuchbar. Ein weiterer, bislang kaum beachteter Vorteil von Texten ist, dass sie sich unabhängig von Umgebungsvariablen prozessieren lassen. Jede Art der Verarbeitung von Texten wollen wir *textprozessierendes Experiment* nennen. Ein solches kann als streng deterministisch angesehen werden: Es ist gleichgültig, welche Temperatur und welcher Luftdruck herrscht, während man das Experiment ausführt. Ebenso ist es gleichgültig, auf welcher Maschine und zu welcher Uhrzeit man arbeitet (jedenfalls, wenn man keine nicht reproduzierbaren zufälligen Variablen in den Programmablauf integriert) und an welchem Ort des Universums man sich gerade befindet. Insofern sind *empirisch-experimentell ausgerichtete textprozessierende Wissenschaften* ein Sonderfall innerhalb der emprischen Wissenschaft: Hypothesen, die auf der Basis von Texten aufgestellt oder untersucht wurden, sind prinzipiell validierbar, da man lediglich das stetige, umgebungsunabhängige, effizient speicher- und damit distribuierbare Ausgangsmaterial (die textuellen Daten) sowie die darauf darauf aufsetzenden textprozessierenden Methoden weitergeben muss, um ein Experiment unter exakt den gleichen Bedingungen zu wiederholen.

2.3 Textwissenschaft als Open Science

Wie lassen sich nun die Vorteile der Textprozessierung nutzen, d.h. wie müssten textprozessierende Wissenschaften betrieben werden, so dass damit die Vorteile, die sich durch das empirische Arbeiten mit stetigen, diskreten, eindimensionalen Daten ergeben, ausgeschöpft werden können? Wenn, wie im letzten Abschnitt dargelegt, textprozessierende Wissenschaften ihre Forschungen im Prinzip auf eine Weise präsentieren können, dass die Ergebnisse jederzeit abrufbar, nachprüfbar und nachvollziehbar werden, dann können womöglich wissenschaftliche Skandale, die auf gefälschten oder fiktiven Ergebnissen beruhen, zumindest für den Bereich der textprozessierenden Wissenschaften ausgeschlossen werden.[33]

33 Publik werden vor allem die Fälschungen von Ergebnissen medizinischer Forschungen. So konnten schon Louis Pasteur, einem der Begründer der Mikrobiologie, durch die posthume Auswertung seiner Tagebuchaufzeichnungen Ungereimtheiten in seinen Forschungen nachgewiesen werden (Geison 1995). Jüngere, Aufsehen erregende Vergehen waren etwa die gefälschten Berichte über die Klonierung menschlicher Stammzellen der koreanischen Forscher Hwang Woo-suk und Park Jong-Hyuk in den Jahren 2004 und 2005 (vgl. Chung *et al.* 2006) oder die Studien des US-amerikanischen

Wesentlich für die wissenschaftliche Forschung ist, dass sie zu intersubjektiv nachprüfbaren und nachvollziehbaren Aussagen über die untersuchten Sachverhalte führen muss. Der Wissenschaftler bearbeitet eine Fragestellung, die sich meist aus vorangehenden Forschungen oder geleisteten Entdeckungen ergibt. Die Fragestellung wird gegebenenfalls in Teilprobleme aufgeteilt, die nacheinander oder auch parallel von verschiedenen Forschern bearbeitet werden können. Wird eins der Teilprobleme gelöst, müssen die Ergebnisse veröffentlicht werden, um sie anderen Wissenschaftlern zugänglich zu machen. Dabei sollte der gesamte Versuchsaufbau wie auch die Messungen sowie deren Auswertungen dargelegt werden. Auch Experimente mit negativem (d.h. nicht erwartetem) Ausgang tragen zum Erkenntnisgewinn der Wissenschaft bei, aus diesem Grund sollten alle Forscher verpflichtet sein, ein Laborbuch zu führen, in dem das tägliche Werk lückenlos dokumentiert wird.[34] Über den gesamten Forschungsprozess wird anschließend ein Manuskript verfasst und bei einem Verlag, einer wissenschaftlichen Zeitschrift oder einer Konferenz zur Veröffentlichung eingereicht. Nach einer Vorprüfung durch den Herausgeber des Mediums reicht dieser das Manuskript an mehrere Gutachter weiter, welche die Darstellung bezüglich ihrer Vollständigkeit, Nachvollziehbarkeit und Korrektheit bewerten. Diesen Prozess nennt man *Peer Review*. Fällt das Urteil positiv aus, so kann die Arbeit veröffentlicht werden und steht prinzipiell anderen Wissenschaftlern zur Verfügung.

Vor dem Aufkommen des elektronischen Publizierens über das Internet bedeutete *prinzipiell zur Verfügung stehen*, dass es jedem Wissenschaftler bzw. jeder wissenschaftlichen Einrichtung offenstand, sich das Medium, über das veröffentlicht wurde, anzuschaffen, also für ein gedrucktes Werk zu bezahlen und es zu inventarisieren. Ab Mitte der 1990er Jahre gingen die Fachverlage dazu über, Zeitschriften auch elektronisch zu publizieren, so

Anästhesisten Scott Reuben, aufgrund derer die Schmerzmittelgabe vor Operationen in Krankenhäusern weltweit beeinflusst wurde. Seit 1996 entpuppten sich mehrere dieser Studien als pure Erfindungen (vgl. `http://www.aaeditor.org/HWP/Retraction.Notice.pdf`, zuletzt aufgerufen am 11.10.2011), so dass das Vertrauen in die medizinische Forschung nachhaltig erschüttert wurde (vgl. `http://idw-online.de/pages/de/news307669`, zuletzt aufgerufen am 11.10.2011).

Weniger beachtet, aber nicht minder zahlreich sind die Fälschungen in anderen Wissenschaften. Eine ständig aktualisierte und sehr umfangreiche, nach Disziplinen geordnete Aufstellung wird u.a. in der deutschsprachigen Wikipedia gepflegt (`http://de.wikipedia.org/wiki/Betrug_und_F%C3%A4lschung_in_der_Wissenschaft`, zuletzt aufgerufen am 11.10.2011), die Zeitschrift *Nature*, die selbst des öfteren gefälschte Studien veröffentlichte, diskutiert Möglichkeiten, wie zukünfig das Vertrauen in die wissenschaftliche Methode wieder hergestellt werden kann (Titus *et al.* 2008).

34 Es wird immer wieder bekannt, dass Studien über die Unwirksamkeit von Medikamenten nicht veröffentlicht werden (vgl. Heinen 2011). Hierzu muss man wissen, dass Wirksamkeit immer nur statistisch nachgewiesen werden kann. Wenn man also dieselbe Studie mehrfach durchführt, steigt naturgemäß die Wahrscheinlichkeit, dass eine der Studien das gewünschte Ergebnis liefert. Informationen zurückzuhalten oder undurchsichtig darzustellen nennt man auch *biased reporting*.

dass man diese – nach Erwerb des zugehörigen Lizenzschlüssels – im Internet betrachten, gegebenenfalls auf den eigenen Rechner herunterladen und – wenn gewünscht – ausdrucken konnte. Etwa zur gleichen Zeit brach etwas aus, was als *Zeitschriftenkrise* bezeichnet wurde. Diese gründete darauf, dass die immer weiter steigenden Zeitschriftenpreise mit rückläufigen bzw. stagnierenden Bibliotheksetats unvereinbar geworden waren. Öffentliche Einrichtungen sahen immer weniger ein, dass sie die Ergebnisse der Forschungen, die sie zum großen Teil voll finanzierten, nach dem Erscheinen in Zeitschriften abermals bezahlen sollten.

Ausgehend von diesem Szenario – Zeitschriftenkrise trifft auf neuartige elektronische Publikationswege – bildete sich eine internationale **Open-Access**-Bewegung, deren zentrale Forderung ist, dass öffentlich geförderte Forschungsergebnisse der Öffentlichkeit auch kostenfrei zur Verfügung gestellt werden müssen. Als Startpunkt der Bewegung wird die Veröffentlichung der Erklärung der *Budapest Open Access Initiative* vom 14.2.2002 angesehen.[35] Inzwischen haben viele Länder basierend auf dieser Erklärung eigene, zum Teil weit darüber hinausgehende Open-Access-Erklärungen formuliert, in Deutschland etwa die *Berliner Erklärung über offenen Zugang zu wissenschaftlichem Wissen* (Oktober 2003)[36], die von allen wichtigen deutschen Forschungsinstitutionen unterschrieben wurde. In dieser Erklärung werden die beiden Voraussetzungen, welche Open-Access-Veröffentlichungen erfüllen müssen, definiert:[37]

1. "Die Urheber und die Rechteinhaber solcher Veröffentlichungen gewähren allen Nutzern unwiderruflich das freie, weltweite Zugangsrecht zu diesen Veröffentlichungen und erlauben ihnen, diese Veröffentlichungen – in jedem beliebigen digitalen Medium und für jeden verantwortbaren Zweck – zu kopieren, zu nutzen, zu verbreiten, zu übertragen und öffentlich wiederzugeben sowie Bearbeitungen davon zu erstellen und zu verbreiten, sofern die Urheberschaft korrekt angegeben wird. (Die Wissen-

35 `http://www.soros.org/openaccess/read.shtml`, zuletzt aufgerufen am 11.10.2011

36 `http://oa.mpg.de/lang/de/berlin-prozess/berliner-erklarung/`, zuletzt aufgerufen am 11.10.2011

37 Die Definition ist der deutschen Wikipedia entnommen, einer Ressource, auf die sowohl per Open Access zugegriffen werden kann und die durch das implizite *Copyleft* durch die entsprechende Creative-Common-Lizenz (hier `http://creativecommons.org/licenses/by-sa/3.0/deed.de`, zuletzt aufgerufen am 11.10.2011) für z.B. diese Arbeit verwendbar ist (und – wenn es nötig gewesen wäre – modifiziert werden könnte). Creative Commons ist eine gemeinnützige Organisation, die Standard-Lizenzverträge entwickelt und veröffentlicht. Ziel ist dabei, dass Autoren der Gemeinschaft großzügige Nutzungsrechte einräumen können (wobei unterschiedliche Freiheitsgrade gewählt werden können). Im Vergleich zu Open-Access- und Open-Science-Initiativen bemüht sich Creative Commons, auf beliebige Werke, deren Schutz sich aus dem Urheberrecht ableiten lässt, anwendbar zu sein.

schaftsgemeinschaft wird, wie schon bisher, auch in Zukunft Regeln hinsichtlich korrekter Urheberangaben und einer verantwortbaren Nutzung von Veröffentlichungen definieren). Weiterhin kann von diesen Beiträgen eine geringe Anzahl von Ausdrucken zum privaten Gebrauch angefertigt werden."

2. "Eine vollständige Fassung der Veröffentlichung sowie aller ergänzenden Materialien, einschließlich einer Kopie der oben erläuterten Rechte wird in einem geeigneten elektronischen Standardformat in mindestens einem Online-Archiv hinterlegt (und damit veröffentlicht), das geeignete technische Standards (wie die Open-Archive-Regeln) verwendet und das von einer wissenschaftlichen Einrichtung, einer wissenschaftlichen Gesellschaft, einer öffentlichen Institution oder einer anderen etablierten Organisation in dem Bestreben betrieben und gepflegt wird, den offenen Zugang, die uneingeschränkte Verbreitung, die Interoperabilität und die langfristige Archivierung zu ermöglichen."

Die Open-Access-Initiative stößt natürlich dort auf Widerstände, wo das althergebrachte Verfahren finanzielle Vorteile gebracht hat, also vor allem bei der Verlagswirtschaft. Diese hat einerseits gut an wissenschaftlichen Zeitschriften verdient, andererseits aber auch durch die Sicherstellung des Peer Review die Praxis wissenschaftlicher Veröffentlichungen jahrzehntelang bestimmt. Auf Peer Review kann auch Open Access nicht verzichten und weil dieser Prozess finanzielle Mittel erfordert, wurden verschiedene Geschäftsmodelle für Open-Access-Veröffentlichungen entwickelt: Entweder kann man die Kosten für eine solche Veröffentlichung dem Autor aufbürden (der diese evtl. an seine Institution, seine Förderorganisation oder seine Auftraggeber weiterleiten kann) oder sie werden von Forschungsverbünden bzw. -institutionen übernommen, die sich durch Mitgliedsbeiträge finanzieren.

So erfolgreich diese wissenschaftliche Praxis in der Vergangenheit war und so vielversprechend die Ansätze zum offenen Zugriff auf Forschungsergebnisse auch sind, hat dieser traditionelle Ansatz, Wissenschaft zu betreiben, doch einen gravierenden Nachteil: Nicht die Forschung selbst wird weitergegeben, nicht die Ausgangsmaterialien, nicht die Versuchsanordnungen werden ausgetauscht, sondern es wird eine Art von Umweg über sprachliche Formulierungen beschritten: Der Forscher muss versuchen, seinen Versuchsaufbau möglichst exakt und widerspruchsfrei in natürlicher Sprache zu umschreiben. Das mag für genormte Methoden der Naturwissenschaft ausreichend sein, im Bereich der Textprozessierung fällt aber auf, wie indirekt dieser Weg ist. Pedersen (2008)[38] zeigt sehr anschaulich,

38 Der Artikel erscheint als *Last Words* der vorletzten gedruckten Ausgabe der Zeitschrift *Computational*

dass – zumindest für virtuelle Wissenschaften – ein solcher Ansatz zu kurz greift und zu sehr unschönen Zuständen führt. *The Sad Tale of the Zigglebottom Tagger* ist eine Parabel über die Diskrepanz zwischen dem Veröffentlichungsdruck und der eigentlich angestrebten Reproduzierbarkeit der Ergebnisse.[39] Pedersen hält es für falsch, dass Empirismus eine Sache des Vertrauens ist, obwohl inzwischen ganz andere Möglichkeiten zur Verfügung ständen:

> "As a community we accept that our publications don't provide enough space to describe our elaborate 21st century empirical methods in sufficient detail to allow for re-implementation and reproduction of results. (...) What's really missing is the software that produced the results that convinced the reviewers the article should be published. This is particularly troubling given the highly empirical nature of the work reported in so many of our publications. We publish page after page of experimental results where apparently small differences determine the perceived value of the work. In this climate, convenient reproduction of results establishes a vital connection between authors and readers." (Pedersen 2008:465)

Pedersen plädiert also dafür, die für die Erlangung der Ergebnisse maßgebliche Software in den Veröffentlichungsprozess einzubinden. Werfen wir hinsichtlich dieser Frage einen Blick auf die **Open-Source**-Initiative für die Publikation von Software, die der Open-Access-Initiative für schriftliche Publikationen vorausgeht. Die Open-Source-Initiative (OSI) wurde 1997 gegründet und war maßgeblich beeinflusst von Raymond (2001), einem Essay, der zur Freigabe des Quelltextes vom *Netscape Navigator* und im weiteren Verlauf zum *Mozilla-Projekt* führte. Schon vorher wurden unterschiedliche Lizenzverträge entwickelt, denen allen gemein ist, dass der Quelltext der Software in einer (für Menschen) verständlichen Form vorliegt, dass die Software beliebig oft kopiert, verbreitet, genutzt und modifiziert werden kann, und dass auch diese Modifikationen distribuiert werden dürfen. Die verschiedenen Lizenzen unterscheiden sich z.B. dadurch, ob die vorgenommenen

Linguistics, die seit 2009 als Open-Access-Magazin ohne Printversion publiziert wird.

39 Die sehr unterhaltsame Erzählung kann grob wie folgt zusammengefasst werden: Der – hypothetische – Zigglebottom-Tagger wird vom Computerlinguisten Zigglebottom in einem – ebenfalls hypothetischen – *Computational-Linguistics*-Artikel als sehr präzise arbeitendes Werkzeug vorgestellt. Leider ist es aber nicht möglich, diese Ergebnisse auf irgendeine Art zu reproduzieren – erst auf wiederholte Anfragen gesteht Zigglebottom ein, dass eine studentische Hilfskraft für die Implementation und die Kalibrierung des Taggers verantwortlich war. Diese ist aber unglücklicherweise inzwischen nicht mehr erreichbar. Eine Programmversion, die Zigglebottom – unvollständig und bar jeder Dokumentation – schließlich doch noch herausgibt, ist völlig unbenutzbar.

Modifikationen wieder unter eine Open-Source-Lizenz gestellt werden müssen (wie z.B. bei der *GNU General Public Licence* – GPL[40]) oder ob dies nicht der Fall sein muss (wie z.B. bei der *Lesser GPL* oder der *Apache Licence*[41]). Letztere sind natürlich auch für die freie Wirtschaft interessant, weil diese damit teilweise auf Open Source basierende, weiterentwickelte Software veräußern kann. Neben der freien Verfügbarkeit bietet der Open-Source-Ansatz auch noch eine Reihe weiterer Vorteile: Die Entwicklung komplexer Programme kann auf eine beinahe beliebig große Anzahl von Personen bzw. Firmen oder Einrichtungen verteilt werden, wodurch jeder von der Arbeit der Anderen profitieren kann. Dadurch können eventuell teure Eigenentwicklungen oder der Kauf von Fremdsoftware vermieden werden. Dazu kommt, dass die Behebung von Programmfehlern oder spezifische Weiterentwicklungen nicht bei einem bestimmten Hersteller beantragt werden müssen, sondern vom Nutzer der Open-Source-Software selbst oder von einem von diesem beauftragten Entwickler vorgenommen werden können. Durch den Einblick in den Sourcecode kann theoretisch jede der entsprechenden Programmiersprache mächtige Person die Softwarequalität analysieren und daraus auf die Wartbarkeit und Maturität der Software schließen. Ähnlich wie bei Open Access ergeben sich auch bei Open Source Fragen nach möglichen Geschäftsmodellen. Open-Source-Anbieter verdienen meist am Support ihrer Produkte oder an Modifikationen, die an spezifische Aufgabenbereiche angepasst und dann als proprietäre Software verkauft werden.[42]

Innerhalb von Science Commons (vgl. Nguyen 2008), einem Unterprojekt von Creative Commons, wird in jüngster Zeit die Etablierung von **Data Papers** vorangetrieben. Diese sollen vor allem dazu dienen, experimentell oder über Messungen gewonnene Daten zu veröffentlichen, um sie der wissenschaftlichen Gemeinschaft zur Verfügung zu stellen:

> "A data paper is a publication whose primary purpose is to expose and describe data, as opposed to analyze and draw conclusions from it. The data paper

40 `http://www.gnu.org/licenses/gpl-3.0.html`, zuletzt aufgerufen am 11.10.2011.

41 `http://www.gnu.org/licenses/lgpl.html` bzw. `http://www.apache.org/licenses/`, beides zuletzt aufgerufen 11.10.2011.

42 Hinsichtlich der Diskussion um Open-Source-Software sei hier noch ein theoretischer Einwand des Informatikers Niklaus Wirth erwähnt: Wirth äußert sich kritisch zur technischen Qualität komplexer Open-Source-Projekte. Die Open-Source-Bewegung ignoriere und behindere die Vorstellung, komplexe Softwaresysteme basierend auf streng hierarchischen Modulen aufzubauen. Entwickler sollten den Sourcecode der von ihnen verwendeten Module nicht kennen. Sie sollten rein auf die Spezifikationen der Schnittstellen der Module vertrauen. Wenn, wie bei Open-Source, der Sourcecode der Module vorhanden ist, führe das automatisch zu einer schlechteren Spezifikation der Schnittstellen, da ja das Verhalten der Module im Sourcecode nachlesbar ist (vgl. `http://www.simple-talk.com/opinion/geek-of-the-week/niklaus-wirth-geek-of-the-week/`, zuletzt aufgerufen am 11.10.2011).

> enables a division of labor in which those possessing the resources and skills can perform the experiments and observations needed to collect potentially interesting data sets, so that many parties, each with a unique background and ability to analyze the data, may make use of it as they see fit." (Rees 2010)

Einer der zentralen Vorteile dieser neuen Publikationsmethode erwächst in der Möglichkeit, durch die Weitergabe von Daten kooperative Forschung betreiben zu können (s.u.), gleichwohl müssen diverse Anforderungen an die Standardisierung, Dokumentation und Reinheit der Daten erfüllt werden, die teilweise erst noch innerhalb der wissenschaftlichen Community ausgehandelt werden müssen.[43]

Open Access, Open Source und Data Papers sind Konzepte, die sich in einen Fundus unterschiedlicher Ansätze aus der Informationstechnologie einfügen, welcher in jüngerer Zeit unter den Namen Cyberinfrastructure oder **e-Science** (übertragen auf die einzelnen Wissenschaftsbereiche e-Geography, e-Humanities, e-Medicine oder e-Engineering) bekannt geworden ist. Obschon noch nicht abschließend geklärt ist, was genau unter e-Science verstanden werden soll oder kann, so ist die Grundidee doch die Stärkung kooperativer Forschung durch die Möglichkeiten moderner Informationstechnologien. Über die oben vorgestellten Konzepte von offenem Zugang zu Wissen und Freigabe des Quellcodes von Software hinaus werden im Bereich e-Science neuere Ansätze zu e-Learning, Wissensmanagement und Grid-Computing[44] bzw. zum Ausbau der Computer-Infrastruktur verfolgt. Auf europäischer Ebene wurde von der Europäischen Union dazu die *ESFRI* (European Strategy Forum on Reasearch Infrastructures)[45] Initiative ins Leben gerufen, die insgesamt 44 Projekte mit 20 Milliarden Euro fördert und sich zum Ziel gesetzt hat, bis 2020 eine europäische Forschungsinfrastruktur aufzubauen. Zwei der Projekte sind dabei für den geistes- und sozialwissenschaftlichen Bereich vorgesehen.[46] Die deutsche

43 Vgl. dazu auch die Ausführungen des Düsseldorfer Linguisten Puschmann unter `http://blog.ynada.com/228`, zuletzt aufgerufen am 11.10.2011.

44 Als Grid wird meist ein virtueller Superrechner angesehen, der aus einer Reihe verbundener Hardware-Ressourcen besteht. Foster (2002) zählt zu den Eigenschaften, die ein Grid in jedem Fall erfüllen sollte, dass es Ressourcen koordiniert, die nicht einer zentralen Instanz untergeordnet sind, dass es offene, standardisierte Protokolle und Schnittstellen verwendet und dass es Dienste bzw. Güter ("nichttrivialer Art") bereitstellt.

45 `http://ec.europa.eu/research/infrastructures/index_en.cfm?pg=esfri`, zuletzt aufgerufen am 11.10.2011.

46 Das Projekt *CLARIN* (Common Language Resources and Technology Infrastructure, `http://www.clarin.eu/external/`) strebt dabei die Harmonisierung struktureller und terminologischer Unterschiede innerhalb einer Grid-Infrastuktur für Sprachdaten an; das Projekt *DARIAH* (The Digital Research Infrastructure for the Arts and Humanities, `http://www.dariah.eu/`, beides zuletzt auf-

D-Grid-Initiative[47] wurde zu dem Zweck gegründet, eine nachhaltige Grid-Forschungsinfrastruktur aufzubauen. Dabei werden einerseits Einzelprojekte für die verschiedenen Wissenschaftsbereiche gefördert (Astronomie im *Astro-Grid*, Klimaforschung im *C3-Grid*, Ingenieurwissenschaften im *INGrid*, Lebenswissenschaften im *MediGrid* und viele mehr), auf der anderen Seite wird aber auch (im D-Grid-Integrationsprojekt) an einer Middleware gearbeitet, welche die einzelnen Grids integrieren soll. Tatsächlich wird auch bereits ein Projekt aus dem Bereich der Geisteswissenschaften, namentlich *TextGrid* gefördert.[48]

Taylor *et al.* (2006) identifizieren die Verwendung von **Workflows** als das zentrale Element im Bereich e-Science. Das Konzept von Workflows wurde zuerst für betriebswirtschaftliche Anwendungsszenarien genutzt, seit den 1990er Jahren werden Referenzmodelle und Standards von der *Workflow Management Coalition* (WfMC)[49] entwickelt. Die WfMC definiert einen Workflow dabei als "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." Dieses Modell wurde auf die e-Science-basierte wissenschaftliche Praxis übertragen, wo schon vorher der Begriff des *in-silico*-Experiments für computerunterstützte wissenschaftliche Tätigkeit konzipiert wurde.[50] Workflows können den Anforderungen an moderne Wissenschaft hinsichtlich Formalisierung und Strukturierung der Datenanalyse gerecht werden, indem sie für folgendes Sorge tragen (Taylor *et al.* 2006:3):

- Komplexe Analysen werden in eine Task-Sequenz zerlegt und in diesem Zuge dokumentiert.
- Die einzelnen Tasks werden (evtl. auf verteilten Ressourcen) ausgeführt.
- Die während der Analyse erzeugten Ergebnisse werden gesammelt und zugänglich gemacht.
- Die Analyse kann gegebenenfalls wiederholt und innerhalb der wissenschaftlichen Community ausgetauscht werden.

Wenn sich über die Gestalt von in-silicio-Experimenten durch die Vereinheitlichung der

gerufen am 11.10.2011) ist ausgerichtet auf die Erstellung und Nutzung von Werkzeugen und die Verflechtung von Bedürfnissen von Informations-Nutzern, -Managern und -Providern.

47 `http://www.d-grid.de/`, zuletzt aufgerufen am 11.10.2011.

48 `http://www.textgrid.de`, zuletzt aufgerufen am 11.10.2011, vgl. dazu auch Schwiebert (2011: Kap. 3.3).

49 The Workflow Management Coalition `http://www.wfmc.org`, zuletzt aufgerufen am 11.10.2011.

50 *In silico* ist eine Wortneuschöpfung, die einen neuen Zweig experimenteller Wissenschaft (neben *in vivo* – Experimente in Lebewesen und *in vitro* – Experimente außerhalb von Lebewesen, z.B. im Reagenzglas) bezeichnet. Der Begriff wurde in den frühen 1990er Jahren im Umfeld der Bioinformatik geprägt, vgl. auch Sieburg (1991).

Erstellung und Repräsentation von Workflows Einigkeit erzielen lässt, so ergeben sich gravierende Vorteile für die wissenschaftliche Praxis, denn "[I]t is much easier to comprehend, providing the opportunity to verify or modify an experiment" (Barga & Gannon 2006:14). Basierend auf diesen Überlegungen definieren De Roure *et al.* (2010) drei Generationen zukünftiger Forschungsumgebungen, die sie **e-Labore** nennen: In der gegenwärtigen ersten Generation werden Werkzeuge (Tools), Daten und Methoden in kleinem Rahmen innerhalb einzelner wissenschaftlicher Disziplinen ausgetauscht, vereinzelt werden auch digitale Artefakte, wie Workflows oder der Zugang zu Analysedaten, publiziert. In der zweiten Generation, die gerade in der wissenschaftlichen Praxis Fuß fasst, gibt es bereits Sammlungen von Daten, Tools und Methoden in Gestalt von eigenständigen, rekombinierbaren und reproduzierbaren Forschungsobjekten, die diziplinenübergreifend eingesetzt werden können. In der zukünftigen dritten Generation von e-Laboren schließlich bestimmt die globale Wiederverwendung ("global reuse") und der fundamentale Austausch ("radical sharing") dieser Forschungsobjekte das Bild der Wissenschaft. Die Autoren sind in der britischen *MyGrid*-Initiative[51] zusammengeschlossen, welche den Weg für diese dritte Generation von e-Laboren bereiten will. MyGrid entwickelt dafür unterschiedliche Werkzeuge, die momentan noch v.a. auf die biowissenschaftliche Community zielen. Mit *Taverna*[52] wurde auch eine virtuelle Umgebung zur Erstellung, Verwaltung und Ausführung von Workflows entwickelt, die über die Biowissenschaften hinaus genutzt werden soll; erwähnt werden Chemie, Sozialstatistik und Music Information Retrieval (vgl. 2.1). Weiterhin wurde mit *MyExperiment*[53] eine Plattform nach dem Vorbild virtueller sozialer Netzwerke geschaffen, welches als eine Art Workflow-Bazaar für unterschiedliche Workflow Management Systeme konzipiert ist (vgl. De Roure *et al.* 2009). Workflows werden damit Gegenstand für die Weitergabe von Forschungsmethoden und darüber hinaus auch für deren Begutachtung im Peer-Review-Prozess.

Ausgehend von der anfänglichen Fragestellung nach den Besonderheiten textprozessierender Wissenschaften wurden hier einige neuere Konzepte für den Informationsaustausch zum Zwecke der kooperativen Forschung aufgeführt: Open Access als Ansatz, Forschungsergebnisse verfügbar zu machen, Open Source als Paradigma für die kooperative Entwicklung und Nutzung von Software, Data Papers zum Zwecke des Austauschs von Rohdaten und schließlich e-Science als Oberbegriff, unter dem sich diese Konzepte bündeln lassen. Einer der interessantesten Gedanken im Bereich e-Science ist, dass durch moderne softwa-

51 `http://www.mygrid.org.uk/`, zuletzt aufgerufen am 11.10.2011.
52 `http://www.taverna.org.uk/`, zuletzt aufgerufen am 11.10.2011, siehe auch Hull *et al.* (2006)
53 `http://www.myexperiment.org/`, zuletzt aufgerufen am 11.10.2011

retechnologische Mittel der Austausch zwischen Wissenschaftlern – über den Transfer von Werkzeugen, Methoden und Ergebnissen – auf einer völlig neuen Ebene realisiert werden kann. Das Hauptargument für die freie Verfügbarkeit von Ergebnissen und Software ist mithin kein finanzielles, sondern ein wissenschaftliches. Wie oben gezeigt, plädiert Pedersen (2008) dafür, Forschungsergebnisse nicht nur über eine natürlichsprachliche schriftliche Fixierung weiterzugeben, sondern auch die Software, mit der diese Ergebnisse gewonnen wurden, zu veröffentlichen. Allein diese kann sicherstellen, dass die schriftlich publizierten Ergebnisse auch reproduziert werden können. Pedersen verlangt, dass die Software ausreichend dokumentiert und einfach zu benutzen sein soll. Was aber macht Software einfach benutzbar? Der Umgang mit komplexen Programmen, die eventuell über keine oder nur eine sehr rudimentäre Benutzerschnittstelle verfügen oder die die Einrichtung weiterer Anwendungen erfordern, stellt computeraffine Wissenschaftler (wie bspw. Leser der Computational Linguistics) möglicherweise vor keine großen Probleme. Dies dürfte aber zumindest für Teile der Forscherschaft, die hier im Fokus stehen und computergestützte, empirisch-experimentelle Textprozessierung bereits betreiben oder gerne betreiben würden, nicht gelten. Dessen ungeachtet ist ein großer Teil der Forschung unmittelbar von der eingesetzten Software abhängig:

Sie wird genutzt, um Daten (Texte) zu verarbeiten und Ergebnisse zu produzieren, nur durch sie kann man folglich diese Ergebnisse auch *re*produzieren. Will man auf die produzierten Daten aufbauen, benötigt man entweder diese Daten oder die Software, um sie wiederholt zu generieren. Will man die Ausgangsdaten verändern, benötigt man die Software, um sie auf diese abweichenden Daten anzuwenden. Will man die Vorgehensweise leicht modifizieren, benötigt man die Software, um diese anzupassen. Will man Methoden fachfremder Forschungen auf die eigenen Daten anwenden, muss man die Software, in der diese Methodik implementiert ist, auf den eigenen Bereich anpassen. Die Software ist damit gleichsam ein *virtueller Arbeitsplatz* für den textprozessierenden Wissenschaftler. Der Untersuchungsgegenstand Text eignet sich – durch seine in 2.2 aufgeführten Eigenschaften – außergewöhnlich gut für Experimente *in silico* und damit auch für die Übernahme des Workflow-Paradigmas, wie es u.a. von der britischen MyGrid-Initiative propagiert wird.

Die zentrale Problemstellung für die kooperative empirisch-experimentelle Textprozessierung mündet also in der Frage nach den Anforderungen an die Software, mit der man diese betreiben will. Diese Anforderungen werden im nächsten Kapitel spezifiziert.

2.4 Zusammenfassung und Überleitung

Aus den bisherigen Überlegungen zu textprozessierenden Wissenschaften, Besonderheiten textueller Daten und einer offener gestaltbaren Textwissenschaft lassen sich zusammenfassend Anforderungen an einen Arbeitsplatz für die Textprozessierung ableiten. Werfen wir zunächst einen Blick auf die etablierte Arbeitsweise der Naturwissenschaften für die experimentelle Arbeit: Hier werden Laboratorien genutzt, in denen Materialien, Geräte und Hilfsmittel, kurz: Eine Infrastruktur für Experimente zur Verfügung stehen. Für Experimente wird eine genau definierte Situation in einer Versuchsanordnung präpariert, welche ebenso wie die zwischenzeitlichen und auch finalen Beobachtungen oder Messungen in einem Laborheft dokumentiert wird, um die Anforderungen an wissenschaftliche Experimente (Nachvollziehbarkeit, Wiederholbarkeit, Objektivität) zu erfüllen. Die skizzierte Software als Arbeitsplatz für die Textprozessierung sollte im Grunde ein solches Labor im virtuellen Sinne bereitstellen. Dazu muss sie eine Möglichkeit bieten, vorhandene (Software-)Werkzeuge und (Software-)Messinstrumente experimentell anzuordnen, um die zu analysierenden Rohmaterialien (Texte) zu verarbeiten. Im Bereich der Softwaretechnologie werden solche Werkzeuge, die bestimmte Teilprobleme lösen und mit anderen Werkzeugen kombiniert werden können, als Komponenten bezeichnet. Der Einsatz von Komponenten erfordert immer eine umgebende Infrastruktur, ein sogenanntes Komponentensystem (vgl. 3.2.1).

Das Verlangen nach der Kombinierbarkeit von Werkzeugen folgt aus dem Sachverhalt, dass Verfahren zur Verarbeitung textueller Daten meist auf vorgelagerten Schritten basieren oder selbst die Basis einer nachgelagerten Verarbeitung bilden. Für die verschiedenen Bereiche der Textprozessierung stehen bereits unterschiedliche elaborierte Softwarelösungen für bestimmte Teilprobleme zur Verfügung. Im Bereich der Sprachverarbeitung etwa existieren Tools für die Tokenisierung von Texten, für die Auszeichnung von Tokens (z.B. hinsichtlich ihrer Wortart), für deren Klassifizierung (etwa als Eigenname), für die Gruppierung von Tokens (etwa in Chunks oder Phrasen).[54] Neben proprietären Lösungen werden auch eine Reihe von integrativen Plattformen entwickelt, die sich am *Software Architecture for Language Engineering* (SALE) Paradigma orientieren, das eine Infra-

54 Umfangreiche Aufstellungen zu sprachverarbeitenden Tools finden sich unter `http://registry.dfki.de/` (zuletzt aufgerufen am 11.10.2011). Diese Sammlung wurde auf Initiative der ACL (Association for Computational Linguistics) angelegt; Ersteller von Softwarelösungen können ihre Tools dort registrieren lassen) und `http://www.uow.edu.au/~dlee/CBLLinks.htm` (zusammengestellt von David Lee, zuletzt aufgerufen am 11.10.2011).

struktur für die maschinelle Sprachverarbeitung beschreibt (für eine Übersicht vgl. Cunningham & Bontcheva 2006). Zu den bekanntesten dieser Systeme zählen *GATE* (General Architecture for Text Engineering)[55] und *UIMA* (Unstructured Information Management Architecture)[56]. In diesem Zusammenhang ist auch das oben erwähnte, in Deutschland entwickelte TextGrid zu nennen. Diese Systeme bieten einerseits eine Art Werkzeugkasten für die textbasierte Arbeit, indem sie die Teilproblem-Lösungen als kombinierbare Komponenten zur Verfügung stellen, andererseits aber auch eine Entwicklungsumgebung für die Implementation und Integration neuer Komponenten. Das Kombinieren von Komponenten wird auch als '*Programmieren im Großen* bezeichnet, da sich der Anwender lediglich mit Schnittstellen, nicht mit Implementationsdetails auseinandersetzen muss, weil über dem tatsächlichen Code eine Abstraktionsschicht liegt. Textprozessierende Komponenten sollten sequentiell genutzt werden können, d.h. der Output einer Komponente sollte als Input für andere Komponenten dienen können.[57] Darüber hinaus sollten Komponenten eine Konfigurationsschnittstelle anbieten, damit sie für verschiedene Anwendungsszenarien (wie etwa die Verarbeitung unterschiedlicher natürlicher Sprachen) bei Bewahrung der Abstraktionsschicht angepasst werden können. Schließlich sollte sie die Nutzung verschiedener Input-Formate und evtl. die Prozessierung im Netz (mit weit größeren Ressourcen, als Arbeitsplatzrechner bieten können) erlauben.

Gesucht wird eine Software, die den Anforderungen kooperativer wissenschaftlicher Textprozessierung genügt. Wie gezeigt wurde, bildet diese Software idealerweise einen laborativen Arbeitsplatz ab, an dem die verschiedensten notwendigen Werkzeuge vorhanden sind oder integriert werden können. Diese Werkzeuge sollten daher – softwaretechnologisch gesehen - als Komponenten realisiert sein, die gesuchte Software ist dementsprechend im Kern ein Komponentensystem. Dieses System sollte so angelegt sein, dass auch große Textmengen prozessiert werden können, es würde eine gewisse Grundausstattung von Werkzeugen, Apparaturen und Messinstrumenten benötigen, die nach Bedarf auch erweitert werden könnte. Das Inventar bestände nicht aus einem monolithischen Block, sondern aus kleinteiligen Komponenten, die miteinander kompatibel sind und zu immer neuen Versuchsaufbauten (Workflows) kombiniert werden können. Diese Komponenten sind dabei idealerweise konfigurierbar, d.h. ihre Arbeitsweise ist innerhalb bestimmter Grenzen mo-

55 `http://www.gate.ac.uk`, zuletzt aufgerufen am 11.10.2011

56 `http://www.research.ibm.com/UIMA`, zuletzt aufgerufen am 11.10.2011

57 Das heißt, dass Komponenten zu Prozessketten (bzw. -graphen) verschaltet werden können. Beispiel für so eine Prozesskette von vertikal verschalteten Komponenten wäre ein Parser, der auf einen Tokenizer und einen Wortarten-Tagger aufsetzt.

difizierbar. Schließlich sollte es ein einem Laborheft vergleichbares Dokumentationssystem geben, mit dem die Versuchsaufbauten über Komponenten und ihren Konfigurationen dokumentiert werden. Damit könnten einerseits Experimente verstetigt, andererseits aber auch auch ein Exportformat für die Weitergabe von Experimenten geboten werden.

Ziel muss sein, ein komponentenbasiertes Softwaresystem bereitzustellen, mit dem es möglich ist, diese Anforderungen umzusetzen. Dabei muss dieses System die folgenden Eigenschaften aufweisen:

1. Damit unterschiedliche textprozessierende Wissenschaften von den integrierten Methoden profitieren können, sollte die Verarbeitung unterschiedlicher Textformate sichergestellt werden. Genomische Daten sollten genauso mit dem System prozessiert werden können wie unterschiedliche natürlichsprachliche Korpora.
2. Experimente sollten auf einfache Art und Weise durch Programmieren im Großen erstellt werden können. Die softwaretechnologische Implementierung sollte vor dem Benutzer verborgen werden.
3. Experimente (bestehend aus Rohdaten, dem Versuchsaufbau und den gewonnenen Ergebnissen) sollten lückenlos dokumentiert werden und einfach veröffentlicht werden können, um kolloborative Wissenschaft zu ermöglichen.[58]

Das nächste Kapitel stellt eine konkrete Instanz einer Arbeitsumgebung vor, die in Bezug auf diese Anforderungen entwickelt wurde: Das *Text Engineering Software Laboratory*, kurz *Tesla*.

58 Ideal wäre ein Arbeitsplatz, der für andere geöffnet werden kann, wie das auch von den Autoren des Data Paper Whitepaper (vgl. 2.3) gefordert wird. Im Bereich der Software-Entwicklung geht z.B. Mylyn (`http://www.eclipse.org/mylyn`, zuletzt aufgerufen am 11.10.2011; vgl. auch Kersten 2007)), ein eclipse-Plugin, mit Hilfe dessen Task-orientierte kollaborative Entwicklung unterstützt wird, diesen Weg.

Our inability to reproduce results leads to a debilitating paradox, where we as reviewers and readers accept highly empirical results on faith. We do this routinely, to the point where we seem to have given up on the idea of being able to reproduce results. This is the natural consequence of faith-based empiricism, and the only way to fight that movement is with a little bit of heresy.

(Ted Pedersen: Empiricism is not a Matter of Faith)

Kapitel 3

Tesla – Ein Labor für Textwissenschaftler

Das System Tesla[1] (Akronym für *Text Engineering Software Laboratory*) wurde an der Sprachlichen Informationsverarbeitung (Institut für Linguistik – Universität zu Köln) entwickelt und ist eine Instanz der in Kapitel 2 skizzierten Arbeitsumgebung für textprozessierende Wissenschaftler. Tesla richtet sich dabei sowohl an Entwickler von Werkzeugen, die in der Textprozessierung eingesetzt werden können, als auch an Anwender, welche diese Werkzeuge miteinander kombinieren, um Analysen auf textuellen Daten durchzuführen. Dieser doppelte Ansatz resultiert in zwei Anwendungen, die dank des Konzepts unterschiedlicher Perspektiven beide auf Basis der Plattform *Eclipse*[2] realisiert werden konnten. Als Perspektive bezeichnet man im Kontext von Eclipse ein Set von Tools bzw. Fenstern, die für eine bestimmte Arbeitsumgebung benötigt werden. Konkret werden in Tesla zwei Arbeitsumgebungen realisiert:

- Die *Developer Perspective* für Entwickler, die eine integrierte Entwicklungsumgebung (IDE) für die Erstellung von Tesla-Werkzeugen (Komponenten) bietet.
- Die *Linguist Perspective* für Anwender von Werkzeugen auf Textdaten, die Funktionalitäten wie Korpus- und Experimentverwaltung sowie einen Workflow-Editor bietet.

Der Fokus dieser Arbeit liegt auf dem Einsatz von Tesla für die Analyse textueller Daten, was innerhalb der zweiten Perspektive realisiert wird. Wichtig ist hier vor allem die Herausstellung zentraler Konzepte, die gewährleisten, dass Tesla als kooperatives Werkzeug für die Textprozessierung eingesetzt werden kann, indem es die in 2.4 spezifizierten An-

1 Zu beziehen ist Tesla unter `http://tesla.spinfo.uni-koeln.de/download.html` (*last stable*) bzw. `http://tesla.spinfo.uni-koeln.de/nightlies/download.html` (*nightly build*), beide zuletzt aufgerufen am 11.10.2011.

2 Ursprünglich startete das Projekt Eclipse, entwickelt von IBM, als IDE für die Programmiersprache Java. Mittlerweile kann das inzwischen quelloffene Tool durch seine seit der Version 3 optimierte Plugin-Struktur als IDE für diverse weitere Programmiersprachen, aber auch als Plattform für jede denkbare Anwendung verwendet werden. Kapitel 3.3.4 geht auf die Vorteile ein, die sich durch die Verwendung von Eclipse ergeben.

forderungen erfüllt; die Darstellung geht deshalb an dieser Stelle nicht auf die Nutzung von Tesla als IDE und weniger auf konkrete Implementierungsdetails ein.[3]

Grundlagen für die Architektur von Tesla waren zum einen die wissenschaftstheoretisch motivierte Forderung nach einer Trennung von empirischen Daten und deren Interpretation (vgl. McEnery 2003), zum anderen die softwaretechnologische Forderung nach der Aufteilung komplexer Aufgaben in Lösungen für Teilprobleme (vgl. Szyperski 1998). Basierend auf diesen Kriterien wurde Tesla als ein Komponentensystem implementiert, bei dem die prozessierenden Komponenten die Ausgangsdaten nicht verändern, sondern sie durch die gewonnenen Informationen anreichern. Dazu nutzt Tesla einen Annotationsgraphen (vgl. Bird & Liberman 1999), der in Datenbanken (`TeslaDBs`[4]) persistiert wird.

Die Textprozessierung stellt unter Umständen umfangreiche Anforderungen an die Hardware, speziell wenn sehr große Datenmengen oder sehr laufzeitintensive oder speicherplatzbedürftige Algorithmen zum Einsatz kommen. Um diesen Anforderungen begegnen zu können, wurde Tesla als mehrschichtiges System implementiert, bei dem Anwender im Client ihre Daten verwalten, Experimente zusammenstellen und die Ergebnisse betrachten können, während der Server die gegebenenfalls aufwendige Prozessierungsarbeit (die Ausführung der Experimente) und die Persistierung der Daten übernimmt. Damit wird vermieden, dass die Ressourcen von Arbeitsplatzrechnern zeitweise oder (im äußersten Fall) vollständig blockiert werden.

Die drei Anforderungen, die im letzten Kapitel mithilfe einer Labormetapher formuliert wurden, zielen auf ein Software-System, das von textprozessierenden Wissenschaftlern als computerbasierter Arbeitsplatz, gleichsam als virtuelles Labor, genutzt werden kann. Die Gliederung dieses Kapitels ergibt sich aus diesen drei zentralen Anforderungen: Kapitel 3.1 stellt die Konzepte vor, die es ermöglichen, Tesla für die Prozessierung sehr unterschiedlicher Textformate (in der Laboranalogie *Rohdaten*) einsetzen zu können. Kapitel 3.2 zeigt, wie in Tesla Experimente realisiert werden können, indem Software-Komponenten kon-

3 Beides wird ausführlich behandelt bei Schwiebert (2011: Kap. 4), des weiteren ist in dieser Hinsicht die Konsultation der Online-Dokumentation mit einer umfangreichen Sammlung von Tutorials zu Tesla unter `http://tesla.spinfo.uni-koeln.de/` (zuletzt aufgerufen am 11.10.2011) zu empfehlen.

4 TeslaDBs bestehen, wie der Plural auch ausdrückt, aus verschiedenen Datenbanken, die über unterschiedliche Persistenzframeworks angesteuert werden. Dazu zählen relationale Datenbanken (RDB, unterstützt werden zur Zeit *Hypersonic* und *postGres*) die über das Persistenzframework Hibernate (`www.hibernate.org`, zuletzt aufgerufen am 11.10.2011) gesteuert werden, Objektdatenbanken (DB4O, `http://www.db4o.com`, zuletzt aufgerufen am 11.10.2011) sowie eigene Entwicklungen (TunguskaDB, orientiert am *Google File System*, vgl. Ghemawat *et al.* 2003) für flach hierarchisierte Daten. Vorteil der Einbindung heterogener Datenbanken ist die Möglichkeit, das Persistenzframework auswählen zu können, das am besten auf die Erfordernisse der zu speichernden Datenstruktur passt.

figuriert und miteinander verknüpft werden (in der Laboranalogie *Versuchsaufbau*). In Kapitel 3.3 wird schließlich dargelegt, wie Tesla die Schaffung einer kooperativen Basis für textprozessierende Wissenschaftler unterstützt. Zunächst werden Experimente – durch die Veröffentlichung des vollständigen Versuchsaufbaus – für die gesamte Community reproduzierbar. Das kooperative Element geht indes über die bloße Reproduktion von Ergebnissen hinaus, da die veröffentlichten Experimente mittels Schnittstellen zu ihrer Konfiguration auch modifizierbar sind. Damit kann ein und derselbe Versuchsaufbau auf unterschiedliche Daten angewendet werden, einzelne Konfigurations-Parameter können abweichend gesetzt, ganze Komponenten ausgetauscht oder neu arrangiert werden. Die Veröffentlichung von Experimenten über das System Tesla impliziert damit neben einer Möglichkeit, Ergebnisse zu überprüfen, auch die Option, eigene Ideen zu Modifikationen des experimentellen Setups schnell und komfortabel umsetzen zu können. Weiterhin wird es durch die Verwendung von konfigurierbaren Komponenten möglich, diese in unterschiedlichen textprozessierenden Wissenschaften einzusetzen, so dass durch sie eine Schnittstelle geschaffen wird, durch die diese wissenschaftlichen Disziplinen miteinander in Kontakt kommen und sich über den Transfer von Methoden gegenseitig bereichern können.

3.1 Ein Labor zur Prozessierung unterschiedlicher Textformate

In diesem Kapitel werden die Konzepte vorgestellt, mit denen Tesla eine generische Schnittstelle für unterschiedliche Texte gewährleistet. Zunächst wird die Verwaltungsschnittstelle für diese Rohdaten in Gestalt des `TeslaCorpusManager` dargestellt (3.1.1). Daran anschließend wird gezeigt, wie sich diese Rohdaten in einzelne Verarbeitungseinheiten aufteilen lassen und wie sie als Dokumente in Tesla repräsentiert sind (3.1.2). Der Zugang zu den Inhalten der Dokumente erfolgt in Tesla über *Reader* (3.1.3), die zugleich als die ersten prozessierenden Komponenten von Experimenten angesehen werden können, da sie die Rohdaten insoweit interpretieren, als dass sie den *Content*[5] von Struktur- und Metainformationen trennen. Der Content wiederum wird erst auf der nächsten Stufe interpretiert, der Präprozessierung (3.1.4), auf der die meisten nachgelagerten Prozessierungsschritte beruhen und die deshalb hier exemplarisch für eine Content-verarbeitende Komponente in Tesla dargestellt wird.

5 Content bezeichnet hier konkrete Daten, die von abstrakten Einheiten unterschieden werden, welche diesen Daten Funktionen zuzuordnen oder diese zu gruppieren vermögen (vgl. Lobin 2001:9).

3.1.1 Die Verwaltung von Texten: Der Tesla CorpusManager

Ein System, das zur Prozessierung von Texten genutzt werden soll, muss zunächst die Verwaltung von textuellen Daten sicherstellen. Programme, die diese Verwaltungsaufgaben übernehmen, werden im Allgemeinen als *Corpus Manager* bezeichnet. Über Art und Umfang der Funktionalitäten, die diese Programme anbieten sollten, besteht indes keine Einigkeit. Unter anderem werden Nutzerverwaltung (welcher Nutzer hat Zugang zu welchen Daten), Suchfunktionalitäten (Konkordanzen, Indizes), Berechnung statistischer Kennwerte (Worthäufigkeiten, Satzlängen) sowie Hinzufügen von Annotationen und Metadaten zu den Funktionalitäten gezählt, die ein Tool zur Korpusverwaltung bieten sollte.[6] Für das hier vorgestellte System zur empirisch-experimentellen Textprozessierung stehen die folgenden Anforderungen im Vordergrund:

1. Als Rohdaten für Experimente kommen beliebige Texte infrage. Demzufolge müssen möglichst alle gängigen Textformate (vgl. Kapitel 2.2) importiert werden können. Die Daten müssen hierbei persistiert werden, damit ihr Zugriff nicht nur zum Zeitpunkt der Ausführung von Experimenten, sondern auch bei deren späterer Überprüfung gewährleistet ist. Im Zuge des Imports sollte das Encoding[7] der Daten – soweit vorhanden – erkannt und Zugang zu ihrem Inhalt geschaffen werden (vgl. 3.1.3).
2. Sehr umfangreiche Ressourcensammlungen, die auf beliebigen Datenträgern (CD-ROM, DVD-ROM, Festplatten, Remote-Medien) vorliegen und für deren Unveränderlichkeit gesorgt ist (etwa das BNC[8] oder die TIGER-Treebank[9]), müssen nicht notwendig abermals persistiert werden. Der Corpus Manager muss auch zu solchen Ressourcen einen Zugang gewährleisten, ohne sie vollständig in die `TeslaDBs` aufnehmen zu müssen.
3. Der Anwender muss auf einfache Weise Textsammlungen zusammenstellen können, die sowohl aus den in die `TeslaDBs` importierten Texten, als auch aus den

6 Ein Überblick zum Funktionsumfang unterschiedlicher bestehender Corpus Manager findet sich in Kouklakis *et al.* (2007).

7 Als Encoding wird die eindeutige Zuordnung von Schriftzeichen (Buchstaben oder Ziffern) und Symbolen innerhalb eines Zeichensatzes bezeichnet.

8 BNC steht für British National Corpus. Diese Textsammlung umfasst 100 Millionen Wörter geschriebener und transkribierter gesprochener Sprache und wurde in den frühen 1990er Jahren an der Universität Oxford zusammengestellt. Durch die Einbeziehung unterschiedlicher Textsorten in verschiedenen Gewichtungen soll ein repräsentativer Querschnitt des britischen Englisch der zweiten Hälfte des 20. Jahrhunderts abgebildet werden.

9 Das Projekt TIGER wird getragen von einer Kooperation der (Computer)Linguistischen Institute der Universitäten Saarbrücken, Stuttgart und Potsdam. Entstanden ist dabei u.a. eine annotierte Baumbank von 50.000 Sätzen mit annähernd einer Millionen Token.

Texten der externen Ressourcen bestehen können. Diese Textsammlungen (in Tesla `DocumentSelections` genannt) sind das Ausgangsmaterial für Experimente (vgl. 3.2.4).

In Tesla übernimmt serverseitig der `TeslaCorpusManager` die Verwaltung von Texten, der Client ist mit einem eigenen View ausgestattet, von dem aus der Anwender die oben aufgelisteten Funktionalitäten zur Verfügung gestellt bekommt: Über den *Upload Document*-Dialog kann der Anwender Verzeichnisse oder einzelne Dokumente referenzieren, die in die `TeslaDBs` importiert werden (Abbildung 3.1). Im Zuge des Imports wird die Codierung und der für die Dokumentenklasse zuständige `Reader` (vgl. 3.1.3) gesetzt. Beides lässt sich manuell modifizieren, dabei kann man im Vorschaufenster (Abbildung 3.2) den Inhalt des Dokuments, den der gewählte Reader mit der eingestellten Codierung liefert, überprüfen.

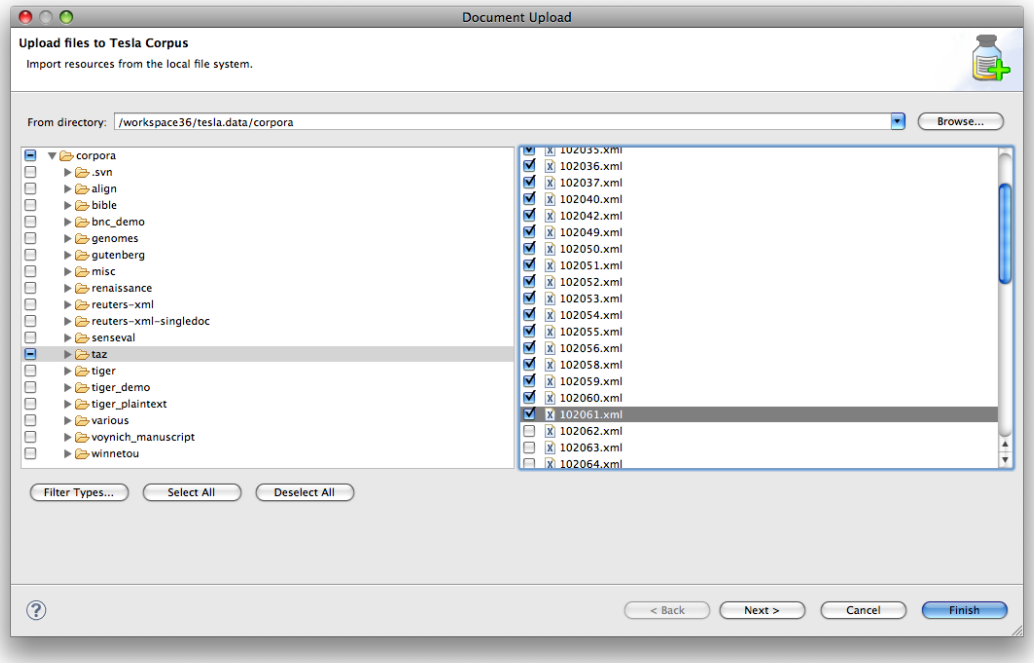


Abbildung 3.1: *Auswahl von Korpusdokumenten. In der Zeile oben wird ein Ordner ausgewählt, dessen Unterordnerstruktur in der linken Spalte dargestellt wird. Aus diesen Unterordnern können nun die zu importierenden Texte ausgewählt werden (rechte Spalte).*

Für den Zugang zu bestehenden Ressourcen gibt es in Tesla die Möglichkeit, `DocumentProvider` einzubinden, welche zur Laufzeit den Inhalt der Dokumente aus diesen Ressourcen liefern. Exemplarisch wurde ein `DocumentProvider` für gepackte Dateien eingebunden

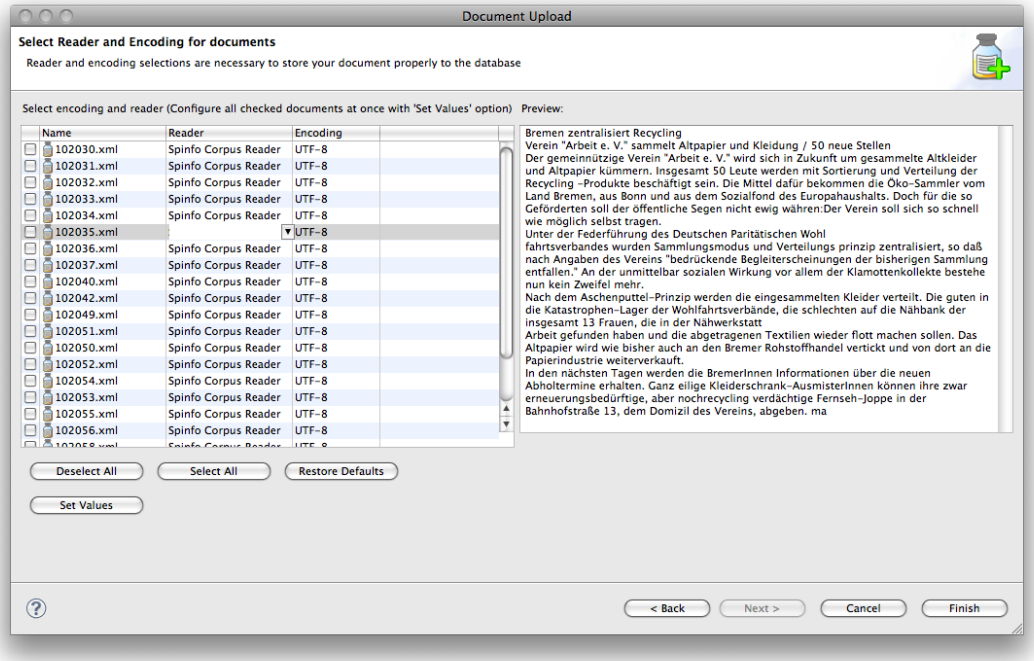


Abbildung 3.2: *Voransicht von Korpusdokumenten. Hier können der automatisch ausgewählte Reader und das Encoding in der Vorschau überprüft (rechte Seite) und gegebenenfalls geändert werden (linke Seite).*

(zip), der z.B. die Texte der gepackten BNC-Datei zugänglich macht. Weitere Ressourcen im zip-Format können erfasst werden, indem sie in der `ExternalCorpora.xml`-Datei, die sich im Client befindet, registriert werden. Zugriff auf abweichende Formate müssen zunächst über einen eigenen `DocumentProvider` realisiert werden.

Beim Upload von Dokumenten wird defaultmäßig eine neue `DocumentSelection` für die gemeinsam hochgeladenen Dokumente erstellt. Will man eigene Textsammlungen erstellen, so kann man dafür im *Create New Selection*-Dialog beliebige Dokumente auswählen. Dabei besteht kein Unterschied zwischen den Dokumenten in den TeslaDBs und denen externer Ressourcen.

Der `TeslaCorpusManager` verfügt auch über eine Reihe von Suchfunktionalitäten, die den Auswahlprozess von Dokumenten unterstützen, etwa um DocumentSelections zu generieren, in denen ausschließlich Dokumente mit bestimmten Metadaten oder Wörtern enthalten sind. Für spätere Versionen ist die Implementation einer Benutzerverwaltung angedacht, die evtl. aus bestehenden *Open Source Tools* zur Verwaltung von Textsammlungen, z.B. aus dem Projekt TextGrid, übernommen werden könnte.

3.1.2 Instanzen von Texten: Das TeslaDocument

Texte werden in Tesla nach dem Import durch den CorpusManager als Objekte des Typs `TeslaDocument` repräsentiert. Kern dieser Klasse ist eine Referenz auf ein `DocumentData`-Objekt, welches den Text in einem Byte-Format zur Verfügung stellt. Diese sehr fundamentale Art der Speicherung erlaubt es, dass textuelle Daten nicht unbedingt als String repräsentiert werden müssen, sondern dass auch andere Formate, wie z.B. MIDI[10] verwendet werden können. Die Schnittstelle zu `DocumentData` ist abstrakt gehalten, um es zu ermöglichen, dass bei großen externen Textkorpora (wie etwa dem BNC) nicht zwangsläufig sämtliche Daten in die `TeslaDBs` importiert werden müssen (vgl. 3.1.1). Wie im vorherigen Kapitel ausgeführt, wird für Referenzen auf `TeslaDBs`-externe Dokumente ein entsprechender `ContentHandler` benötigt, der dafür sorgt, dass auf `DocumentData`-Objekte, die in Experimenten benötigt werden, zugegriffen werden kann.

`TeslaDocuments` verfügen als Datenbankobjekte über eine ID, mit der sie unmissverständlich zum einen in den `TeslaDBs` – aber auch darüber hinaus (vgl. 3.3.1) – eindeutig referenziert werden können. Ferner enthalten die Dokumente Informationen darüber, zu welchem Korpus sie gehören (d.h., ob sie in den `TeslaDBs` oder in externen Ressourcen zu finden sind), in welchen Selektionen sie referenziert sind, welches Encoding sie haben und schließlich welcher `TeslaReader` imstande ist, ihren Inhalt zu interpretieren (siehe 3.1.3). Diese Informationen können über den `Properties`-View eingesehen werden (Abbildung 3.3).

3.1.3 Der Zugang zu Texten: Die Tesla Reader

Wie im vorangehenden Kapitel gezeigt, gewährleistet die Speicherung von Texten in einem Byte-Array, dass sämtliche Arten von Daten, die sich derart repräsentieren lassen (das wären etwa Zeichenfolgen, aber auch Dateien im MIDI-Format), als Rohdaten für die experimentelle Analyse in Tesla ausgewählt werden können. Um diese Rohdaten weiter zu verarbeiten, müssen sie interpretiert werden. Diese Interpretation kann auf mehreren Stufen erfolgen; zunächst einmal muss der tatsächliche Inhalt (Content, s.o.) des Textes von Format- und Metainformationen getrennt werden. Diese Aufgabe wird in Tesla von sogenannten *Readern* übernommen. Es wurden bereits eine Reihe von Readern für unterschiedliche Datenformate implementiert (vgl. Anhang C.1). `Reader` sind als generischer

10 Englisch für *musical instrument digital interface*, ein Datenübertragungs-Protokoll für musikalische Steuerinformationen.

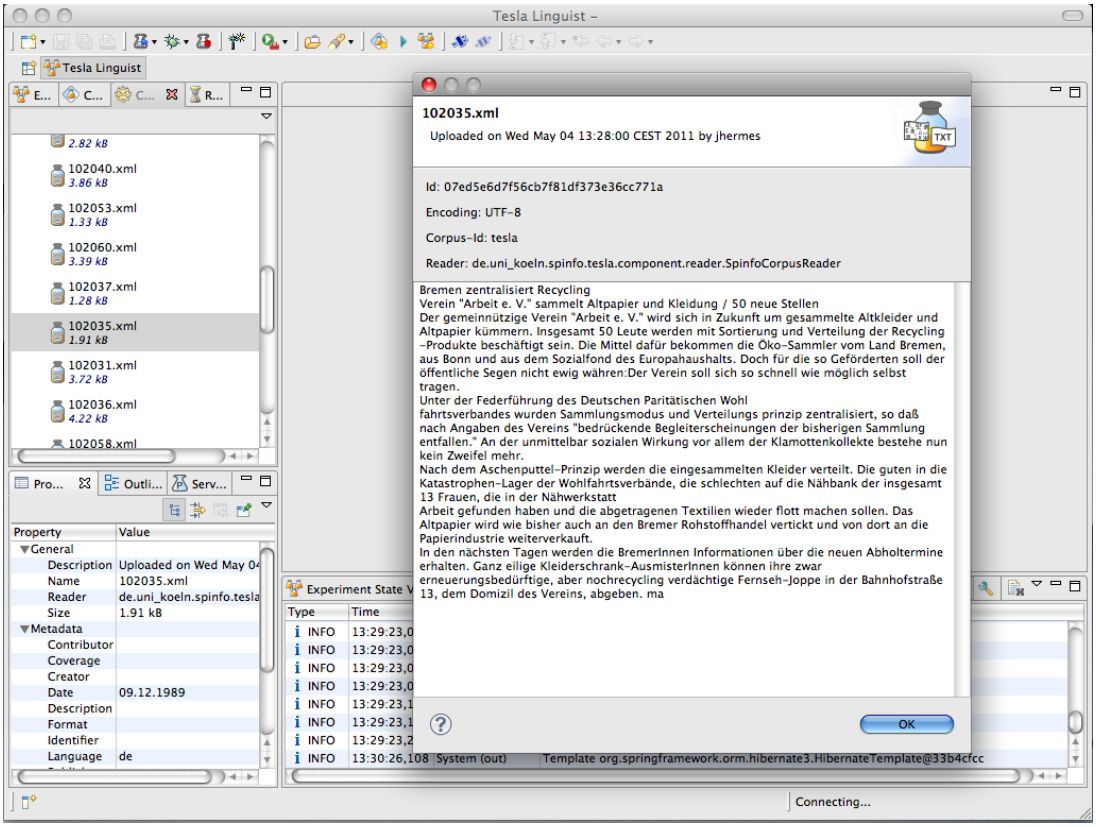


Abbildung 3.3: *Auf der linken Seite sieht man den CorpusManager-View, in dem die bestehenden Selections mit den enthaltenen Dokumenten aufgelistet sind. Die Properties angewählter Dokumente kann man im Properties-View unten betrachten. Per Doppelklick auf ein Dokument öffnet sich der Document-View, in dem sämtliche Attribute sowie der Inhalt des ausgewählten Dokuments dargestellt werden (Fenster auf der rechten Seite).*

Typ implementiert und liefern in jedem Fall den Content der Rohdaten für den für sie bestimmten Typ (`Reader` vom Typ `String` liefern also Strings, MIDI-`Reader` liefern parallele MIDI-Spuren usw.).

Bei der bisherigen Entwicklung von Readern standen solche im Vordergrund, die als Content Strings liefern. Dabei wurden unterschiedlich spezifische Reader implementiert. Der generellste von ihnen ist der `TextReader`, der einfach sämtliche Bytes des `TeslaDocuments` auf einen String schreibt – für den `TextReader` ist also das gesamte Rohdatum Content. Da er sämtliche `String`-basierten Formate behandeln kann, ist der `TextReader` der Default-Reader für derartige Daten. Um einen Grad spezialisierter sind Reader, welche die Ausgangsdaten insoweit interpretieren, als dass sie den eigentlichen Inhalt des Dokuments von Format- und Metainformationen trennen, etwa `XMLReader` und `HTMLReader`, die als Content

lediglich den Inhalt der Dokumente liefern und sämtliche Element-Auszeichnungen tilgen. Die Reader der nächsten Spezialisierungsstufe vermögen auch diese Auszeichnungen zu interpretieren und sie als `Annotationen`[11] zum Ausgangstext in der Datenbank abzulegen. Sie können allerdings nur bestimmte, streng definierte Formate prozessieren, der `SpinfoCorpusReader` etwa nur Dokumente im SpinfoCorpus-Format (vgl. Abbildung 3.1), wobei er Paragraphengrenzen und eine Untermenge der *Dublin-Core*[12]-Metadaten als Annotationen schreibt; der `BNCReader` benötigt BNC-kompatible Daten und erfüllt damit auch Aufgaben, die normalerweise durch Tokenizer und POS-Tagger geleistet werden, indem er Annotationen für Tokens und deren POS-Tags generiert und den Zugriff auf diese Informationen bereitstellt (vgl. Kapitel 3.1.4). Reader auf dieser Spezialisierungsstufe können also auch die Aufgaben von Komponenten übernehmen.[13]

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <text date="09.12.1989" type="newspaper" source="taz" domain="misc"
      language="de">
3    <paragraph>\&quot;Aqua-Mix\&quot;</paragraph>
4    <paragraph>Vorschaltger&aumlt f&uumlr die Waschmaschine spart
         Strom</paragraph>
5    <paragraph>Die Kiste ist grau, misst etwa 10x20 cm, hat vier Schalter,
         zwei L&aumlmpchen, drei Wasseranschl&uumlsse und einen
         Elektroanschluss.
6    </paragraph>
7 </text>
```

Listing 3.1: *Ausschnitt aus einer Datei im SpinfoCorpus-Format. Es handelt sich um ein einfach gehaltenes Format, in das unterschiedliche Zeitungsarchive (Taz, FAZ Natur und Wissenschaft, Süddeutsche) konvertiert wurden, um einen relativ hürdenlosen Zugang zu diesen Archiven zu erhalten. Die Texte sind in Paragraphen unterteilt und mit Meta-Informationen angereichert, die einer Untermenge des Dublin Core entsprechen (Datum, Quelle etc.). Aufgabe des für dieses Format entworfenen SpinfoCorpusReader ist es, diese Format- und Metainformationen in Annotationen zu überführen und den Content als String zu liefern.*

Beim Upload von neuen Dokumenten durch den `CorpusManager` können die Reader für diese Dokumente explizit gesetzt werden, was aber – vor allem beim Upload sehr großer

11 Zum Konzept der Tesla-`Annotationen` vgl. 3.2.1.

12 Der Dublin Core ist eine Sammlung einfacher und standardisierter Konventionen zur Beschreibung von Dokumenten und anderen Objekten im Internet, um diese mit Hilfe von Metadaten einfacher auffindbar zu machen. Urheber dieses Schemas ist die „Dublin Core Metadata Initiative“ (DCMI), vgl. `http://dublincore.org/`, zuletzt aufgerufen am 11.10.2011.

13 Zu Komponenten und dem Tesla-Rollenkonzept vgl. 3.2.2. Basierend auf dem *Apache Tika Project* wurden darüber hinaus eine Reihe von Readern entwickelt, welche die gängigsten Textformate (pdf, odf, doc etc.) interpretieren können (vgl. `http://tika.apache.org/`, zuletzt aufgerufen am 11.10.2011).

Textmengen – einen hohen manuellen Aufwand erfordert. Daher bietet Tesla eine automatische Reader-Auswahl an, wobei der spezifischste Reader ausgewählt wird, der das Dokumentenformat unterstützt.[14]

Tesla Reader verfügen über zwei Schnittstellen (spezifiziert im `TeslaReader`-Interface), über die der `CorpusManager` mit ihnen kommuniziert:

- `getPreview(InputStream i, OutputStream o, String encoding)` schreibt den Content des übergebenen InputStream auf den übergebenen OutputStream. Der Reader prozessiert dabei in seiner Default-Konfiguration, Annotationen werden nicht erzeugt.
- `supportsContent(InputStream input)` liefert `true`, wenn der Reader das Dokumentenformat auf dem übergebenen InputStream lesen kann, `false` im gegenteiligen Fall.

Die Ausführung von Readern innerhalb von Experimenten erfolgt – wie bei Komponenten, siehe 3.2.1 – über eine Methode, die mit einer `@Run`-Annotation versehen ist. Dabei stellt das Tesla-Laufzeitsystem sicher, dass die entsprechenden Experiment-Konfigurationen sowie die Input- und Output-Adapter des Readers gesetzt sind. Reader sind insofern technisch gesehen spezialisierte Tesla-Komponenten.[15] Speziell sind Reader deswegen, weil sie über einen `SignalInputAdapter` als Input- und einen `SignalOutputAdapter` als Output-Schnittstelle verfügen müssen. Über die Input-Schnittstelle erhält der Reader Zugriff auf den Byte-Stream des zu verarbeitenden Textes, dessen Content er über seine Output-Schnittstelle UTF-8-codiert den Text-konsumierenden Komponenten zur Verfügung stellt. Weiterhin unterscheiden sich Reader auf der konzeptuellen Ebene von anderen Komponenten dadurch, dass sie lediglich Struktur- und Metainformationen von Text-Entitäten auswerten und evtl. in Annotationen zum Text überführen. So ermitteln sie zwar den Content von Text-Entitäten und stellen ihn bereit, interpretieren ihn aber nicht weiter. Die Aufgabe dieser Content-Interpretation muss daher von anderen Komponenten übernommen werden, von denen im nächsten Kapitel expemplarisch ein Präprozessierer vorgestellt wird.

14 Dabei werden alle verfügbaren Reader-Klassen mit absteigender Spezifizität über die `supportsContent(InputStream input)`-Methode befragt. Der erste Reader, der meldet, dass er das Dokumentenformat unterstützt, wird ausgewählt. Für String-basierte Formate wird spätestens der `TextFileReader` eine positive Rückmeldung geben, weshalb dieser als Default-Reader für solcherlei Daten gilt, s.o.

15 Dies drückt sich auch in der Klassenhierarchie aus: Tesla-Komponenten erben alle aus der abstrakten Klasse `TeslaComponent`, von der auch die Oberklasse der Reader, `TeslaReaderComponent` abgeleitet ist.

3.1.4 Der erste Schritt zur Interpretation von Texten: Präprozessierung

Als *Präprozessor* werden Programme bezeichnet, die Eingabedaten aufbereiten und zur weiteren Verarbeitung an andere Programme weitergeben. Sie finden sich häufig im Bereich der Programmiersprachen, wo sie vom Compiler für den ersten Übersetzungsschritt aufgerufen werden (etwa bei der Programmiersprache C) oder als Generatoren von spezifischen Formaten dienen – so können etwa XSLT-Programme als Präprozessoren für XML-Dateien angesehen werden. Im Bereich des Text Mining unterscheiden Feldman & Sanger (2006) zwischen *preprocessing tasks* (z.B. Kategorisierung oder Feature- und Term-Extraktion), welche Rohdaten für die weitere Analyse aufarbeiten und *core mining operations* (z.B. Musteridentifikation, Inhaltsanalyse), welche zur Ableitung neuen Wissens aus den Daten führen.

Die Art und Weise der Präprozessierung von Texten hat Einfluss auf alle weiteren Schritte der textprozessierenden Analyse. Wie etwa wird definiert, was ein Wort ist: Werden Wörter, die mit Bindestrich verbunden sind, getrennt behandelt oder als Einheit? Wie wird mit Abkürzungen verfahren, die auf einen Punkt enden? Werden sie ignoriert, könnte der abschließende Punkt fälschlicherweise als Satzendzeichen interpretiert werden. Auch die Behandlung von Sonderzeichen fällt in den Bereich der Präprozessierung. Sie können z.B. normalisiert werden, indem sie durch Zeichenkombinationen aus dem ASCII 128 substituiert werden. Die vielfältigen Probleme, die bei der Tokenisierung und weiteren Vorverarbeitungsschritten auftreten, werden u.a. bei Mikheev (2003) und Hagenbruch (2010) behandelt. Der Einfluss der Präprozessierung mag für viele Analysen, die im Bereich der Textprozessierung durchgeführt werden, gering sein. Die genaue Verfahrensweise präprozessierender Komponenten gehört gleichwohl wesentlich zum Design von textprozessierenden Experimenten und muss wie alle weiteren Schritte auch lückenlos dokumentiert werden, um diese Experimente nachvollziehbar zu halten.

Im günstigsten Fall sind die verwendeten Ressourcen bereits präprozessiert und die in diesem Prozess identifizierte Information ist explizit in ihnen codiert, wie z.B. beim BNC, wo die Rohdaten tokenisiert, mit ihren POS-Tag annotiert und im XML-Format persistiert wurden (vgl. Abbildung 3.2).

Dies gilt aber nur für eine sehr eingeschränkte Menge von Daten, die von großen Universitäten, Forschungsverbünden oder wissenschaftlichen Gesellschaften vorverarbeitet, zur Verfügung gestellt und kontinuierlich gepflegt werden. Dieses Material macht lediglich einen Bruchteil aller verfügbaren Texte aus und kann daher auch nur für einen Bruch-

```
<bncDoc xml:id="A02">
<teiHeader> . . . </teiHeader>
<wtext type="OTHERPUB">
   <div level="1"><head type ="MAIN"><p>
   <s n="21">
      <w c5="AT0" hw="the"    pos="ART">  The      </w>
      <w c5="ORD" hw="last"   pos="ADJ">  last     </w>
      <w c5="CRD" hw="12"     pos="ADJ">  12       </w>
      <w c5="NN2" hw="month"  pos="SUBST">months   </w>
      <w c5="VHB" hw="have"   pos="VERB"> have     </w>
      <w c5="VVN" hw="see"    pos="VERB"> seen     </w>
      <w c5="AJC" hw="far"    pos="ADJ">  further  </w>
      <w c5="AJ0" hw="dramat" pos="ADJ">  dramatic </w>
      <w c5="NN1" hw="growth" pos="SUBST">growth   </w>
      <c c5="PUN">                  .           </c>
   </s>
   </p></head></div>
</wtext>
</bncDoc>
```

Listing 3.2: *Ausschnitt aus einer BNC-Datei. Der Text ist bereits in Paragraphen (p), Sätze (s) und Wörter (w) tokenisiert. Jedes Wort enthält darüber hinaus Informationen zum Lemma (hw) sowie zwei POS-Tags, der erste aus dem C5-Tagset (c5), welches aus 57 verschiedenen Elementen besteht, der zweite aus einem eingeschränkterem Tagset (pos). Die Guideline für BNC-Tagsets findet sich unter http://www.natcorp.ox.ac.uk/docs/URG/posguide.html, zuletzt aufgerufen am 11.10.2011.*

teil der möglichen Analysen verwendet werden. Im Normalfall wird der textprozessierende Wissenschaftler also mit nicht vorprozessierten, d.h. mit nicht annotierten Daten konfrontiert. Dies muss nicht unbedingt ein Nachteil sein, da die Verwendung von annotierten Korpora auch eine Reihe von Problemen mit sich bringt: Je nachdem, ob die Daten maschinell oder manuell ausgezeichnet wurden, weisen sie entweder eine Reihe falscher oder eine Reihe inkonsistenter Klassifikationen auf (vgl. Sinclair 1992). Speichert man die Daten mitsamt dieser Auszeichnungen, so führt dies zu einer Verstetigung dieser Fehler bzw. Inkonsistenzen. Darüber hinaus bedingt Korpusannotation auch immer eine unrevidierbare (sprach)theoretische Entscheidung.[16]

Aus diesen Gründen wurde vorgeschlagen, Annotation von Texten nicht statisch, also als

16 Zwar wird in der Korpuslinguistik immer gefordert, die Daten möglichst *theorieneutral* (vgl. z.B. Leech 1993:275ff) auszuzeichnen, echte Theorieneutralität dürfte jedoch kaum umzusetzen sein. Dies lässt sich z.B. an der Zahl der verschiedenen Tagsets ablesen, die von den derzeit gebräuchlichen POS-Taggern genutzt werden. Wie Abb. 3.2 zeigt, zeichnet allein das BNC seine Wörter mit *zwei* verschiedenen Tags aus zwei verschiedenen Sets aus. Grundlage für die meisten dieser Tagsets bilden die gemeinhin als klassisch angesehenen Wortartenklassen (Nomen, Verben, Adjektive, Präpositionen etc.), die durch morphosyntaktische, teilweise aber auch durch distributionelle Eigenschaften ermittelt werden. Die Klassifizierung von Wörtern kann auch durch reine Distributionsanalyse erreicht werden (dies wird auch ein Thema des Analysen-Kapitels sein, vgl. 6.1), woraus allerdings eine viel größere Zahl von Wortklassen resultiert, als die traditionelle Wortartenlehre annimmt.

persistierte Daten, sondern dynamisch umzusetzen, also als *Teil des Prozessierens* textueller Daten zu betrachten (Benden & Hermes 2004). Dies hat den Vorteil, dass der Anwender selbst auswählen kann, mit welchen Informationen und auf Grundlage welcher (Sprach)Theorie der Text angereichert werden soll. Außerdem werden die Auszeichnungen dadurch nachvollziehbar, da die annotierende Komponente Teil des textprozessierenden Experiments wird. Unterschiedliche Anforderungen verlangen auch unterschiedlich ausgezeichnete Daten. Eine Komponente für dynamische Annotation muss demnach über eine hochkonfigurierbare Schnittstelle verfügen, über welche festgelegt werden kann, wie die Komponente arbeitet. Eine solche Konfigurationsschnittstelle sollte idealerweise die folgenden Bedingungen erfüllen (vgl. Hermes & Benden 2005):

- Die Konfiguration muss **konsequent** sein, insofern die gesamte Arbeitsweise der Komponente manipulierbar ist.
- Die Konfiguration muss **selbsterklärend** sein, indem sprechendes Vokabular für Klassifikationen und Operationen verwendet wird; in der Anordnung wird der Prozessablauf reflektiert.
- Die Konfiguration muss **abstrakt** sein, sie sollte ohne Kenntnisse spezifischer Programmiersprachen begreif- und manipulierbar sein.

Als Referenzimplementation, die eine solche Konfigurationsschnittstelle bietet, kann die präprozessierende Komponente `SPre` angesehen werden. Mit `SPre` können Texte auf verschiedenen Ebenen – (in `SPre Layer` genannt, bspw. Zeichen-, Wort-, Satz- und Paragraph-Layer) tokenisiert und klassifiziert werden.[17] Dabei erfolgt die Angabe von Regeln für diese Operationen innerhalb einer XML-Konfigurationsdatei, die dann von `SPre` interpretiert und umgesetzt wird. Auf jeder Ebene werden valide Einheiten definiert und Klassen zugeordnet. Ausschnitte aus der SPre-Konfiguration eines `CharacterParsers` (der valide Zeichen definiert und klassifiziert) finden sich in der Abbildung 3.3, die etwas komplexere Definition eines Wortes in der WordParser-Konfiguration wurde aus Platzgründen in den Anhang D verschoben.[18]

17 Damit übernimmt der Präprozessor gleich mehrere Rollen linguistischer Analyse: Er ist sowohl ein Wort-Tokenizer, wie auch ein Satzgrenzen-Erkenner, wie auch ein Paragraph-Marker (vgl. Kapitel 3.2.2 zu linguistischen Rollen). Die ausführliche Dokumentation zu SPre findet sich unter `http://www.spinfo.phil-fak.uni-koeln.de/spinfo-forschung-spre.html`, zuletzt aufgerufen am 11.10.2011. Frühe Versionen des Präprozessors können als eine Art Vorstufe des Tesla-Systems betrachtet werden, da `SPre` ursprünglich als Tokenizer angelegt wurde, der auch eine Plattform für annotierende Komponenten bot. Inzwischen wird `SPre` allerdings als reiner Tokenizer (gleichwohl auf verschiedenen Ebenen) genutzt, da das Zusammenspiel mit Annotatoren innerhalb des Tesla-Komponentenmodells erfolgt.

18 Wie unschwer zu erkennen ist, geht der Anspruch von SPre *konsequent konfigurierbar* zu sein einher

```
<spre:characterParser>
 <spre:tokens>
  <spre:token name="a_min">a</spre:token>
  <spre:token name="b_min">b</spre:token> ...
 </spre:tokens>
 <spre:tokenClasses>
  <spre:tokenClass name="LowerCaseLetter">
   <spre:item>a_min</spre:item>
   <spre:item>b_min</spre:item> ...
  </spre:tokenClass> ...
  <spre:tokenClass name="Letter">
   <spre:item>LowerCaseLetter</spre:item>
   <spre:item>CapitalLetter</spre:item>
  </spre:tokenClass>
 </spre:tokenClasses>
<spre:characterParser>
```

Listing 3.3: *Ausschnitte aus einer SPre-Konfigurationsdatei mit beispielhaften Definition valider Einheiten (tokens) und deren Klassifikation (tokenClasses) auf dem CharacterLayer*

Die Einführung von SPre als tokenisierende Komponente im Tesla-System zeigt beispielhaft auf, wie Komponenten in das Framework von Tesla integriert werden müssen, um mit weiteren Komponenten interagieren zu können. Das nächste Kapitel liefert die konzeptuellen Grundlagen, die dieser Komponenteninteraktion zugrunde liegen.

3.2 Ein Labor zur Konfiguration von Experimenten

Experimente lassen sich, gleichgültig, ob in einem realen oder in einem virtuellen Labor durchgeführt, durch die Spezifikation ihrer Ausgangsdaten und ihres zugrundeliegenden Versuchsaufbaus definieren. Für den Versuchsaufbau eines Experiments in der Textprozessierung müssen Werkzeuge zur Verfügung stehen, die unterschiedliche Teilaufgaben verrichten: Texte sequenzieren (Tokenisierer), mit Informationen anreichern (Annotatoren, z.B. ein PoS-Tagger), Sequenzen typisieren (Indexierer, Gazetteer etc.), in neue Strukturen aggregieren (etwa durch statistische Analysen oder Clusterer) oder aus dem Ausgangsmaterial neue Texte erzeugen (wie Zusammenfassung von Texten, Maschinelle Übersetzung). Software-Werkzeuge, die mit anderen Software-Werkzeugen kombiniert werden können, bezeichnet man üblicherweise als *Komponenten*: “Software components are executable units of independent production, acquisition, and deployment that can be composed into a functioning system” (Szyperski 2002:3). Der größte Vorteil in der Verwendung

mit sehr komplexen Konfigurationsdateien. Aus diesem Grund ist SPre nicht der einzige in Tesla verfügbare Tokenizer, es existiert z.B. auch noch die Komponente `SimpleTokenizer`, die nur in sehr viel eingeschränkterem Maße konfigurierbar ist, damit aber auch leichter zu bedienen.

von Komponenten besteht darin, dass diese innerhalb verschiedener Kontexte verwendbar sind, so dass sich einerseits die Produktivität in der Anlage von Software erhöht (da nicht alles immer wieder neu programmiert werden muss), andererseits aber auch die Qualität gesteigert wird, weil Software durch die vielfältigen Anwendungen in verschiedenen Kontexten verbessert wird. Die Erstellung der Software kann damit arbeitsteilig erfolgen, da die Gesamtfunktionalität in einzelne Komponenten-Funktionalitäten gekapselt ist, die durch unterschiedliche Entwickler in Code umgesetzt werden können. Durch die Modularisierung erhöht sich der Abstraktionsgrad; nicht relevante Details werden innerhalb von Komponenten gekapselt und sind für den kooperativen Programmierer wie auch für die Verwender (Kompositoren) der Komponenten nicht sichtbar. Diese Abstraktion kann es auch ermöglichen, Komponenten innerhalb verschiedener Anwendungsszenarien, gegebenenfalls auch über bestehende Grenzen hinweg, z.B. zwischen unterschiedlichen wissenschaftlichen Disziplinen, auszutauschen. Nach Zwintzscher (2004) weisen Komponenten die folgenden Eigenschaften auf:

- Sie haben die Fähigkeit zur Selbstbeschreibung (was ist die Funktionsweise der Komponente?).
- Sie weisen lediglich explizite Kontextabhängigkeiten auf (welche Umgebungsvoraussetzungen müssen erfüllt sein, damit die Komponente korrekt funktioniert?).
- Sie besitzen idealerweise die Fähigkeit zur Anpassung an unterschiedliche Anwendungsszenarien (Adaptierbarkeit).

Für die Komposition von Komponenten zu komplexen Anwendungen ist es notwendig, dass diese klar definierte Schnittstellen aufweisen. Die Schnittstellen ergeben sich aus dem der Entwicklung zugrundeliegenden Komponentenmodell, welches sich wiederum aus den Anforderungen der Komponentenplattform ergibt. Schnittstellen können als eine Art Vertrag zwischen dem Komponentenersteller und dem Anwender angesehen werden, da sie nach der Publikation der Komponente nicht mehr geändert werden sollten.

In diesem Kapitel wird gezeigt, welches die Grundlagen des Tesla-Komponentensystems sind. Es gliedert sich wie folgt: Wie Komponenten realisiert werden müssen, um innerhalb der Komponentenplattform Tesla verwendet werden zu können, wird in 3.2.1 spezifiziert. Den Problemen, die sich aus zwei entgegengesetzten Anforderungen an die Komponentenschnittstellen ergeben, wird mit dem Tesla-Rollensystem begegnet, auf das in 3.2.2 eingegangen wird. Die Anpassungsfähigkeit bzw. die Modellierung expliziter Kontextabhängigkeiten wird durch die externe Konfiguration von Komponenten realisiert (3.2.3). Im letzten Abschnitt 3.2.4 wird die Komposition von Komponenten zu Tesla-Experimenten demonstriert.

3.2.1 Verteilte Arbeit: Tesla Komponenten

Das Paradigma der komponentenbasierten Softwareentwicklung ist aus der modernen Softwaretechnologie inzwischen nicht mehr wegzudenken. So sind z.B. aktuelle Systemarchitekturen für Applikationsserver[19] als skalierbare Containersysteme für Softwarekomponenten angelegt. Diese Systeme basieren auf unterschiedlichen allgemein einsetzbaren Komponentenmodellen, die verbreitesten Standards sind das CORBA (Common Object Request Broker Architecture) Components Model, das DCOM (Distributed Component Object Model) von Microsoft sowie EJB (Enterprise Java Beans) von SUN/Oracle. Außerdem ist in diesem Zusammenhang OSGi (Open Service Gateway Initiative) zu erwähnen, eine dynamische, hardwareunabhängige Softwareplattform, die auf der *Java Virtual Machine* (JVM) aufsetzt und es erlaubt, Dienste per Komponentenmodell zu modularisieren.

Tesla ist serverseitig auf Grundlage des *Spring2*-Frameworks[20], clientseitig als *Eclipse*-Plugin realisiert, wobei die Plugin-Technologie von Eclipse auf *Equinox*, einem OSGi-kompatiblen Framework basiert. Tesla realisiert ein speziell auf die Anforderungen der Textprozessierung angelegtes Komponentenmodell, das die sukzessive Anreicherung von textuellen Daten mit expliziten Informationen ermöglicht.

Das Hauptproblem beim Design von Komponentenmodellen ist die Gewährleistung der Komponenteninteraktion. Textprozessierende Systeme sind zumindest zum Teil sequenziell organisiert, das heißt, dass die Komponenten nicht unabhängig voneinander spezielle Teilprobleme bearbeiten, sondern dass sie auf den Ergebnissen anderer, bereits abgearbeiteter Komponenten aufbauen. Eine PoS-Tagger-Komponente etwa kann erst ihre Arbeit aufnehmen, wenn eine Tokenizer-Komponente die zu taggenden Einheiten sequenziert hat.

19 Der Begriff Applikationsserver wird für Software verwendet, die eine Reihe spezieller Dienste über definierte Schnittstellen zur Verfügung stellt. Ein Applikationsserver bietet eine Laufzeitumgebung für den Server-Teil einer Client-Server-Anwendung (vgl. 3.3.4) und stellt u.a. Dienste für Authentifizierungen, Transaktionen, Verzeichnis- und Datenbankzugriffe zur Verfügung. Die meisten Applikationsserver sind für die Java Enterprise Edition (JEE) entwickelt worden und OpenSource. Zu ihnen zählen als die bekanntesten Vertreter *Apache Geronimo* (kommerzielle Variante: *IBM Websphere*), *Glassfish* (von Oracle) und der *JBoss Application Server*.

20 Das Spring-Framework entstand ursprünglich als Reaktion auf das extrem komplexe Programmiermodell, das man mit den EJB-Versionen bis 2.1 nutzen musste. Die EJB-3-Version nahm sehr viele Anregungen auf, die auf Spring zurückzuführen sind, und bietet dadurch inzwischen auch ein vereinfachtes Modell. Spring hat allerdings weiterhin den Vorteil, dass es sich auch außerhalb von schwergewichtigen Applikationsservern nutzen lässt. Es basiert auf den Prinzipien *Dependency Injection* (auch Hollywood-Prinzip genannt – "Don't call us, we call you" – Objekten werden die benötigten Ressourcen zugewiesen, sie müssen sich also nicht selbst um die Bereitstellung kümmern) sowie *Aspekt-orientierte Programmierung* (AOP, dabei werden klar definierbare Aufgaben wie bspw. Transaktionen und Sicherheit zentral und einmalig behandelt). Eine ausführliche Dokumentation des Frameworks findet sich unter `http://www.springsource.org/documentation`, zuletzt aufgerufen am 11.10.2011.

Die Ergebnisse des PoS-Taggers können dann wiederum notwendige Voraussetzung für die Arbeit einer Chunk-Parser[21]-Komponente sein.

Aufgrund dieser Abhängigkeiten der Komponenten untereinander ist es zweckmäßig, die von ihnen produzierten Ergebnisse als Schnittstelle zu wählen, welche die Interaktion zwischen Komponenten zur Verfügung stellt. Genau so wird dies auch in den meisten bisher entwickelten linguistischen Komponentenframeworks (vgl. 2.4) gehandhabt. Als Schnittstelle kann dabei ein Annotationsgraph dienen. Der Begriff geht auf das ATLAS-Framework (Bird & Liberman 1999) zurück, auf das sich die Entwickler dieser Komponentenframeworks beziehen.[22] Einen Annotationsgraph (der meist als *Directed Acyclic Graph*, kurz DAG realisiert wird) kann man sich als eine Sammlung von Annotationen vorstellen, die über Anker an das zugrundeliegende Signal (den Text) geknüpft sind. Im Vergleich zu OHCO[23]-Formaten wie XML besitzen DAGs den Vorteil, dass überlappende Strukturen abgebildet werden können, die bei der Auszeichnung von Sprachdaten immer wieder auftreten. Dies ist etwa der Fall bei gleichzeitiger Auszeichnung von Silben- und Morphemstrukturen in Wörtern. Es gibt sowohl Silben, die sich über Morphemgrenzen erstrecken als auch Morpheme, die sich über Silbengrenzen erstrecken. Ein Beispiel sei hier das Wort *Segler*, das nach Silben in *Seg-ler*, nach Morphemen aber in *Segl-er* unterteilt wird. Die Morphemgrenze befindet sich damit innerhalb einer Silbe, wie auch die Silbengrenze innerhalb eines Morphems auftritt; die beiden Labels lassen sich nicht in einer Hierarchie abbilden.[24] Eine solche Hierarchisierung ist aber für die Darstellung in einem XML-Format notwendig, die XML-Struktur scheitert deswegen (Abbildung 3.4, linke Seite). Die Darstellung in einem Graphen verlangt indes keine Hierarchisierung, da Kanten beliebig verankert werden können – im Ausgangstext genauso wie an weiteren Kanten. Morpheme und Silben können widerspruchsfrei in einer Struktur dargestellt werden (Abbildung 3.4 rechte Seite).

21 Im Vergleich zu vollständigen Parsern, die einem Satz eine Gesamtstruktur zuordnen, zeichnen Chunk-Parser (auch partielle Parser genannt) nur bestimmte Segmente des Satzes als Konstituenten (Chunks) aus, die bestimmte zu verarbeitende Informationen enthalten. Diese Chunks überlappen sich nicht und sind nicht rekursiv aufgebaut. Die resultierende Struktur ist daher auch flach, nicht hierarchisch. Typischerweise werden nominale, verbale, adjektivische und präpositionale Phrasen als Chunks ausgezeichnet. Eine Vielzahl von Anwendungen verlangt lediglich die Auszeichnung von Nominal-Chunks (vgl. Jurafsky & Martin 2008:484f).

22 Die Idee findet sich auch schon früher, etwa beim Texteditor Lara (Gutknecht 1985).

23 OHCO steht für *Ordered Hierarchy of Content Objects*, also ein streng hierarchisches Format, welches sich als Baum repräsentieren lässt.

24 Es gibt eine Reihe weiterer nicht-hierarchisierbarer (Meta-)sprachlicher Einheiten – z.B. bei gleichzeitiger Auszeichnung von Sätzen und wörtlicher Rede: “Die Fußnote” bemerkte er “ist eigentlich unsinnig. Aber weglassen hat auch nicht mehr Sinn.”

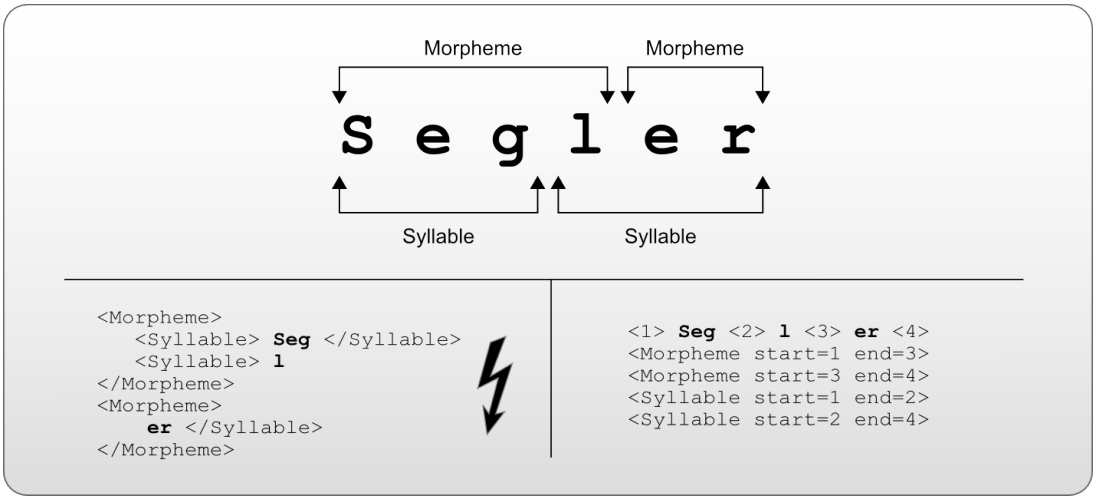


Abbildung 3.4: *Visualisierung des Problems nicht-hierarchisierbarer Entitäten: Scheiternde XML-Struktur (links) und valides DAG-Format (rechts).*

Ein wichtiges Merkmal für Annotationsgraphen sind die Typen von Annotationen, die in diesen aufgenommen werden können. ATLAS und GATE beschränkten sich hier anfangs auf einfache Datentypen, in UIMA kann man immerhin eigene Datentypen entwerfen, die allerdings letztlich aus Kombinationen einfacher Datentypen bestehen müssen.[25] Das schließt die Erzeugung sehr komplexer Datenstrukturen aus. Graphen oder Karten etwa, wie man sie bspw. für die Darstellung von Clustern benötigt, können nicht ohne weiteres in ein interpretierbares Set von einfachen Datentypen konvertiert werden. Ohnehin führt der Ansatz, die von Komponenten produzierten Ergebnisse als Schnittstelle zu wählen, zu zwei gegensätzlichen Anforderungen an diese Schnittstelle:[26] Aus der Sicht des Frameworks sollte die Schnittstelle so restriktiv wie möglich gestaltet werden, um die weitestmögliche Interaktion von Komponenten zu gewährleisten. Aus Sicht der Komponente sollte die Schnittstelle sehr frei sein, damit aussagekräftige Ergebnisse produziert werden können. Während in den oben erwähnten linguistischen Komponenten-Frameworks die Schnittstelle eher restriktiv angelegt sind, bemüht sich der Tesla-Ansatz um ein möglichst flexibles Interface: Im Grunde kann alles, was sich mit Hilfe von Konstrukten der Programmiersprache Java abbilden lässt, als Daten-Container einer Annotation gewählt werden. Wie die Interaktion von Komponenten dennoch – durch die Einführung eines Rollensys-

25 Verschiedene Typen werden dabei in einem *UIMA Type System* definiert. Ein System für linguistische Typen wurde bspw. am Language & Information Engineering Lab der Universität Jena entworfen (vgl. `http://www.julielab.de/Resources/Software/NLP+Tools/Download/UIMA+Type+System-p-98.html`, zuletzt aufgerufen am 11.10.2011).

26 Ausführlich dargestellt wird dieser Konflikt in Götz & Suhre (2004).

tems – gewährleistet werden kann, wird im nächsten Kapitel (3.2.2) ausgeführt.[27]

Realisiert werden Tesla-Komponenten durch spezielle Java-Klassen, welche die Schnittstellen-beschreibenden Metadaten in Form von Java-Annotationen tragen.[28] Eine Tesla-Komponente muss von der Klasse `TeslaComponent` abgeleitet sein und die Annotation `@Component` tragen, die dafür sorgt, dass die Komponente beim Server registriert wird. Die `@Component`-Annotation enthält (als Attribute) überdies noch Informationen über die Komponente, etwa die Namen der Autoren, die Version der Komponente, die Wiederverwendbarkeit ihrer Ergebnisse (vgl. 3.3.3). Eine Methode der Komponentenklasse muss mit @Run annotiertet sein, damit Tesla klar ist, was genau ausgeführt werden soll, wenn eine Komponente aus einem Experiment heraus aufgerufen wird.

Die Ein- und Ausgabeschnittstellen der Komponenten sind über Adapter realisiert, die mit spezifischen Datenobjekten typisiert sind. Diese Datenobjekte sind die Ergebnisse der Komponente, und können – wie oben schon bemerkt – jedwede Java-Klasse sein, solange sie aus der Klasse `DataObject` abgeleitet wurde und damit einer Entity entsprechen, d.h. serialisierbar sind. Diese DataObjects werden mit Annotationen[29] assoziiert, über die sie in den Annotationsgraphen, die grundlegende Austauschstruktur zwischen Tesla-Komponenten, eingebunden werden. Die Klasse `Annotation` dient also dazu, eine gemeinsame Schnittstelle zu definieren, die eine einfache Weitergabe der Ergebnisse ermöglicht, während die von der Klasse `DataObject` abgeleiteten Klassen spezialisierte Formate einzelner Komponentenergebnisse realisieren können. `Annotation` und `DataObject` sind nicht über eine direkte Referenz, sondern über einen gemeinsamen Identifier miteinander verknüpft. Ein Vorteil dieser Konstruktion ist, dass bei Abruf einer großen Zahl von Annotationen die betreffenden DataObjects – erst wenn sie zwingend benötigt werden – zur Laufzeit nachgeladen werden können. Außerdem ist es damit möglich, Querverweise

27 Eine umfassendere Darstellung der Unterschiede, welche die einzelnen Frameworks bezüglich der Restriktionen dieser Schnittstelle machen, findet sich bei Schwiebert (2011: Kap. 3.4).

28 Java-Annotationen sind ein Konstrukt, das im *Java Specification Request* (JSR) 175 festgelegt und mit der Java-Version 5.0 eingeführt wurde. Damit ist es möglich, Meta-Informationen, die vom Compiler oder vom Programm selbst über Reflection (Abfrage von Informationen über Klassen und Instanzen zur Laufzeit) ausgewertet werden können, direkt im Quelltext abzulegen. Separate Beschreibungsdateien (wie z.B. noch in der EJB 2.1-Spezifikation obligatorisch gefordert waren) werden damit überflüssig. Der Code wird übersichtlicher, sämtliche Daten können in einer Klasse hinterlegt und müssen nicht auf mehrere Dateien verteilt werden, was die Wartbarkeit von Programmen extrem erleichtert. Annotationen können darüber hinaus als Beitrag zum *literate programming* (vgl. Knuth 1984) gesehen werden, da sie die annotierten Quellcode-Elemente ortsnah (und nicht extern) dokumentieren.

29 Um Verwechslungen vorzubeugen: Es handelt sich hier um Objekte der Klasse `de.uni_koeln.spinfo.tesla.runtime.persistence.Annotation`, nicht um eine Java-Annotation, die Meta-Informationen mit Java-Sprachkonstrukten verknüpft, s.o.

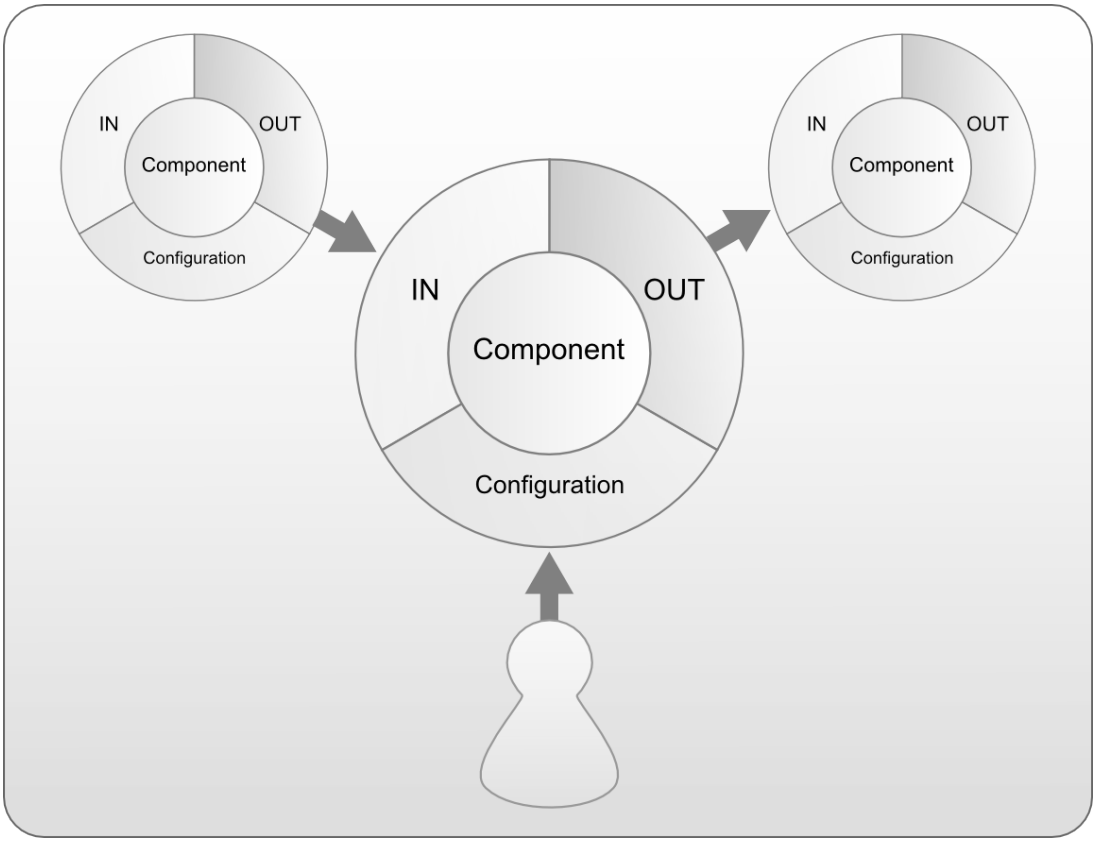


Abbildung 3.5: *Tesla-Komponenten und ihre Schnittstellen: Die Kommunikation mit anderen Komponenten wird mit Input- bzw. Output-Adaptern realisiert, der Anwender kann die Arbeitsweise der Komponente über die Konfigurationsschnittstelle manipulieren.*

zwischen Annotationen, die von verschiedenen Persistenzframeworks verwaltet werden, zu realisieren (ausführlich dargelegt in Schwiebert 2011: Kap. 4.2.2).

Zugriffsmethoden auf die DataObjects selbst sind entweder innerhalb ihrer selbst realisiert (wenn es sich um Zugriffe innerhalb eines einzelnen DataObjects handelt) oder (wenn auf mehrere DataObjects zugegriffen werden soll, indem etwa alle DataObjects desselben Typs angesprochen werden sollen) in nach den Datenobjekten typisierten Adaptern. Komponenten greifen auf solche typisierten Adapter zurück, um Zugriff auf Texte (über `SignalAdapter`) und Annotationen sowie DataObjects anderer Komponenten (über `AccessAdapter`) zu erlangen. Weiterhin stellen sie ihre Ergebnisse über eigene AccessAdapter anderen Komponenten und der Tesla-Visualisierungsschnittstelle zur Verfügung.

Für die Adaption der Komponente an verschiedene Anforderungsszenarien besitzt diese Konfigurationsschnittstellen, die durch Felder und auch Methoden der Komponente realisiert werden können. Dazu wird an die entsprechenden Felder/Methoden der Komponen-

tenklasse eine `@Configuration`-Annotation geschrieben (siehe 3.2.3). Darüber hinaus sind DataObjects und AccessAdapter in Tesla in *Rollen* zusammengefasst (siehe 3.2.2). Abstrakt gesehen handelt es sich bei einer Tesla-Komponente um gekapselte Text-Verarbeitungsalgorithmen, welche Schnittstellen zu anderen Komponenten (Input-Schnittstelle für den Zugriff auf die produzierten Ergebnisse bereits prozessierter Komponenten, Output-Schnittstelle für die produzierten Ergebnisse der Komponente selbst) sowie über die Konfiguration eine Schnittstelle zum Anwender bietet. Visualisiert ist dieses Schema einer Tesla-Komponente in Abbildung 3.5; Listing 3.4 zeigt das Schema als vereinfachten Java-Code.

```
1 @Component(threadMode=ThreadMode.NOT_SUPPORTED, author=@Author(author="John
      Doe", email="jdoe@spinfo.uni-koeln.de",
      web="http://www.spinfo.uni-koeln.de/jdoh", organization="Spinfo"),
      description=@Description(name="ExampleComp", licence=Licence.LGPL_2,
      summary="An example component.",  bigO="linear", version="1.0"))
2 public class ExampleComponent extends TeslaComponent {
3
4     @AccessAdapter(role="ExampleInputRole", name="ExampleInput")
5     private ExampleAccessAdapter<ExampleInputDO> exampleInput;
6
7     @OutputAdapter(dataObject=ExampleDO.class, name="ExampleOut",
          type=ExampleOA.class, accessAdapterImpl=ExampleAA.class)
8     @RoleDescription(value="ExampleOuputRole")
9     private IOutputAdapter<MultiValueCategory> outputAdapter;
10
11    @Configuration(editor="ExampleEditor", defaultValue="true",
          name="Example", min=1, max=1, description="Example CI")
12    private boolean exampleConfig = true;
13
14    @Run(threadMode=ThreadMode.THREAD_PER_SIGNAL)
15    public Result run() throws Exception { ... }
16 }
```

Listing 3.4: *Schema einer Tesla-Komponente mit Komponentenbeschreibung in der @Component-Annotation, Schnittstellen zu Input (AccessAdapter), Output (OutputAdapter, RoleDescription) und Nutzer (Configuration). Innerhalb der Run-Methode wird die eigentliche algorithmische Tätigkeit der Komponente gekapselt.*

Tesla-Komponenten registrieren sich durch ihre `@Component`-Annotation automatisch am Server. Dieser reicht eine Aufstellung über alle registrierten Komponenten an den Client weiter, der sie im Components-View darstellt. Die gegenwärtige Tesla-Distribution verfügt über mehr als 35 Komponenten, darunter alle bisher erwähnten. Letztere werden auch in Anhang C kurz in ihrem Funktionsumfang dargestellt.

3.2.2 Klassifikation textprozessierender Aufgaben: Das Tesla-Rollensystem

Komponenten unterliegen hinsichtlich der Ergebnisse, die von ihnen produziert werden können, keinerlei Restriktionen. Hauptanforderung an Komponentensysteme ist aber, dass die Schnittstellen einer Komponente sowohl deren Wiederverwendung als auch deren Austauschbarkeit gewährleisten sollen, vgl. Hamlet *et al.* (2001:3): "The central dilemma of software design using components is that component developers cannot know how their components will be used and so cannot describe component properties for an unknown, arbitrary situation."

Werden also Framework und Komponentenschnittstelle nicht durch globale Restriktionen eingeschränkt, muss diese Schnittstelle auf eine andere Weise limitiert werden. Eine Möglichkeit wird durch van Gurp & Bosch (2002:5f) aufgezeigt, welche die Einführung lokaler Beschränkungen durch vordefinierte Rollen vorschlagen: "By limiting the communication between two components by providing a smaller interface, the communication becomes more specific. Because of the smaller interfaces the communication also becomes easier to generalize (i.e. to apply a wider range of components). These two properties make the components both more versatile and reusable". Ein solches Rollensystem wurde in Tesla in Form des *Tesla Role System* (TRS) als Basis für die Interaktion textprozessierender Komponenten umgesetzt. Die Grundzüge des TRS werden in den folgenden Abschnitten erläutert.

Angewendet auf textprozessierende Komponenten bedeutet der Entwurf eines Rollenkonzepts die Klassifikation textprozessierender Aufgaben: Komponenten etwa, die den Ausgangstext in kleinere Sequenzen (Tokens) aufsplitten, tragen die Rolle *Tokenizer*. Komponenten, die solchen Tokens PoS-Tags zuweisen, tragen die Rolle *Tagger*; die Rolle *Indexer* wird von Komponenten übernommen, welche Tokens zu Types aggregieren; *Clusterer* berechnen Relationen zwischen Tokens oder Types und geben an, welche einander ähnlicher, welche sich unähnlicher sind. Die Reihe ließe sich beliebig fortführen, das Prinzip bleibt stets das gleiche: Rollen klassifizieren Komponenten gemäß ihren *Ergebnissen*, also anhand der Informationen, welche diese produzieren.

Bei der Vorstellung des Tesla-Komponentenkonzepts (Kapitel 3.2.1) wurde dargelegt, dass Tesla-Komponenten sowohl Input- als auch Output-Interfaces bieten müssen, um mit anderen Komponenten verschaltet werden zu können. Durch die Übernahme einer Rolle legt die Komponente fest, von welchem Typ ihr Output-Interface ist. Die genaue Arbeitsweise der Komponente, wie auch die von ihr vorausgesetzten Input-Daten, werden von der Rolle

nicht erfasst. Rollen klassifizieren Komponenten also nur anhand ihres Outputs, während sie vom Input und dem der Komponente zugrundeliegenden Algorithmus abstrahieren.

Die Output-Schnittstelle einer Komponente besteht, wie bereits oben erwähnt, aus zwei Elementen:

1. Den real produzierten Daten (*DataObjects*), die als Objekte von Annotationen in den Ergebnisgraphen geschrieben werden und Zugriff auf sich selbst bieten.
2. Den Zugriffsmethoden, gekapselt in *AccessAdaptern*, welche durch DataObjects typisiert sind und den Zugriff auf Mengen von Annotationen und diesen zugehörigen DataObjects bereitstellen.

Um eine solche zweiteilige Output-Schnittstelle vollständig definieren zu können, muss eine Tesla-Rolle also sowohl mit einem `DataObject` als auch mit einem `AccessAdapter` assoziiert sein. Diese beiden assoziierten Elemente sind als Interfaces angelegt, eine Rolle ist mithin keine konkrete Implementierung. Dies hat auch den Vorteil, dass die Implementationen der Klassen hinter den Interfaces ausgetauscht werden können, um verschiedene Persistenzframeworks oder Lizenzmodelle nutzen zu können. Rollen selbst sind in Tesla nicht als Interfaces realisiert, sondern liegen als XML-Definitionen vor. Die Definition einer Rolle entspricht dabei dem folgenden Schema:

```
 1 <role id="de.uni_koeln.spinfo.tesla.roles.ExampleRole">
 2  <metadata>
 3   <name>Example Role</name>
 4   <description>Example definition of a role.</description>
 5  </metadata>
 6  <produces>
 7   <analysis>
 8    <data_object_interface>ExampleDataObject</data_object_interface>
 9    <adapter_interface>ExampleAccessAdapter</adapter_interface>
10   </analysis>
11  </produces>
12 </role>
```

Listing 3.5: *Schema einer Tesla-Rollendefinition*

Eine Tesla-Komponente, welche eine bestimmte Rolle realisiert, muss infolgedessen auch eine Realisierung des von dieser Rolle geforderten DataObject-Interfaces produzieren sowie einen AccessAdapter anbieten, der eine Realisierung des von der Rolle geforderten AccessAdapter-Interfaces ist. Der Zusammenhang zwischen Rollen, Komponenten, DataObjects und AccessAdaptern wird in Abbildung 3.6 dargestellt.

Mit der Übernahme einer Rolle durch eine Tesla-Komponente wird lediglich die Output-Schnittstelle festgelegt. Die noch fehlende Input-Schnittstelle wird in der Komponente

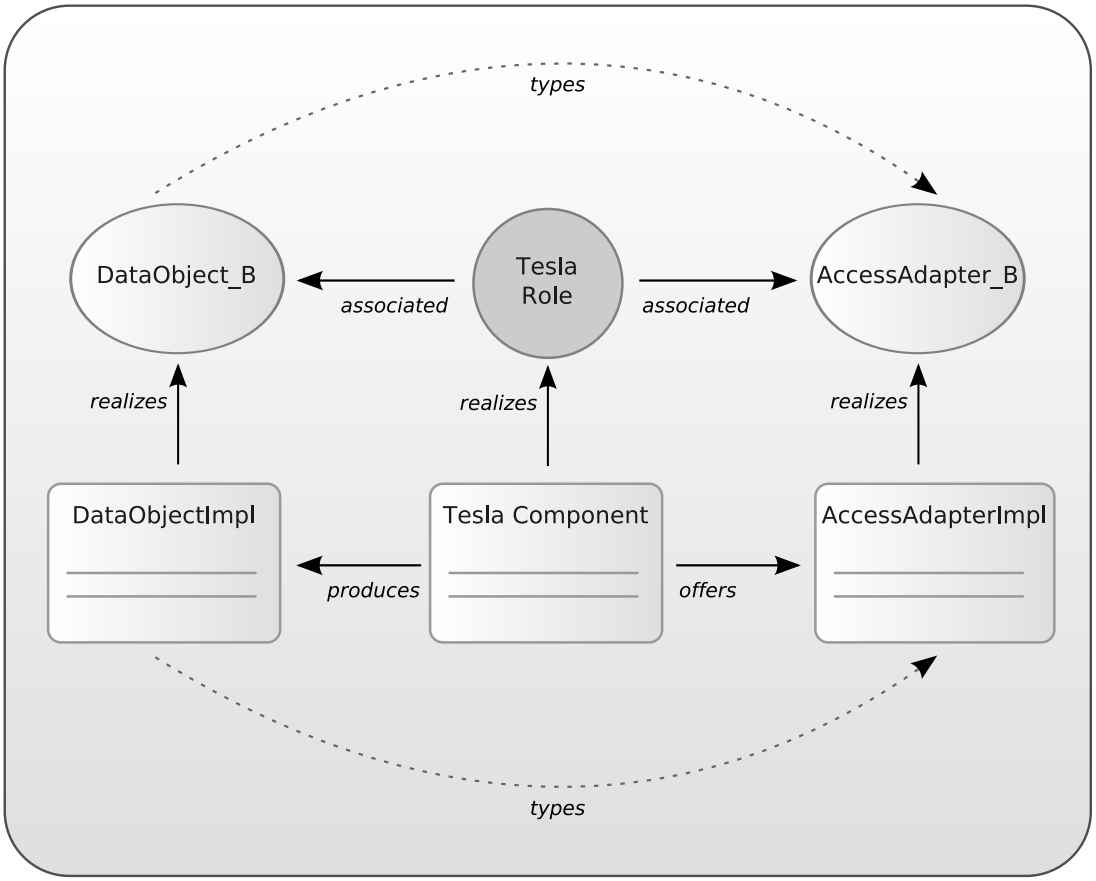


Abbildung 3.6: *Der Zusammenhang zwischen Komponenten, Rollen, Datenobjekten und Zugriffsadaptern. Wenn Komponenten Rollen realisieren, so müssen sie die Implementationen eines DataObjects und eines AccessAdapters bieten, welche die in den Rollen geforderten Schnittstellen realisieren. Abbildung übernommen aus Hermes & Schwiebert (2010:290).*

selbst gesetzt, indem diese eine Rolle (mit zugehörigem DataObject und AccessAdapter) konsumiert. Dies wird in Abbildung 3.7 visualisiert. Die Abbildung zeigt außerdem die Abstraktionsschicht in der Komponenteninteraktion: Komponenten interagieren nicht mit anderen – konkret implementierten – Komponenten, sondern mit Abstraktionen von diesen, nämlich Rollen. Für eine Komponente spielt es schließlich keine Rolle, welches spezielle Bauteil mit welcher Implementierung und aufgrund welcher Datenlage für ihren Input verantwortlich ist, es zählt für sie nur der Input selbst. Die globale Beschränkung auf Komponentenschnittstellen kann gerade deshalb aufgehoben werden, weil Rollen mit lokalen Beschränkungen gewährleisten, dass Komponenten die in den Rollen definierten Schnittstellen einhalten. Eine Rollensammlung kann demnach als eine Registry lokaler Beschränkungen der Komponentenschnittstellen gelten.

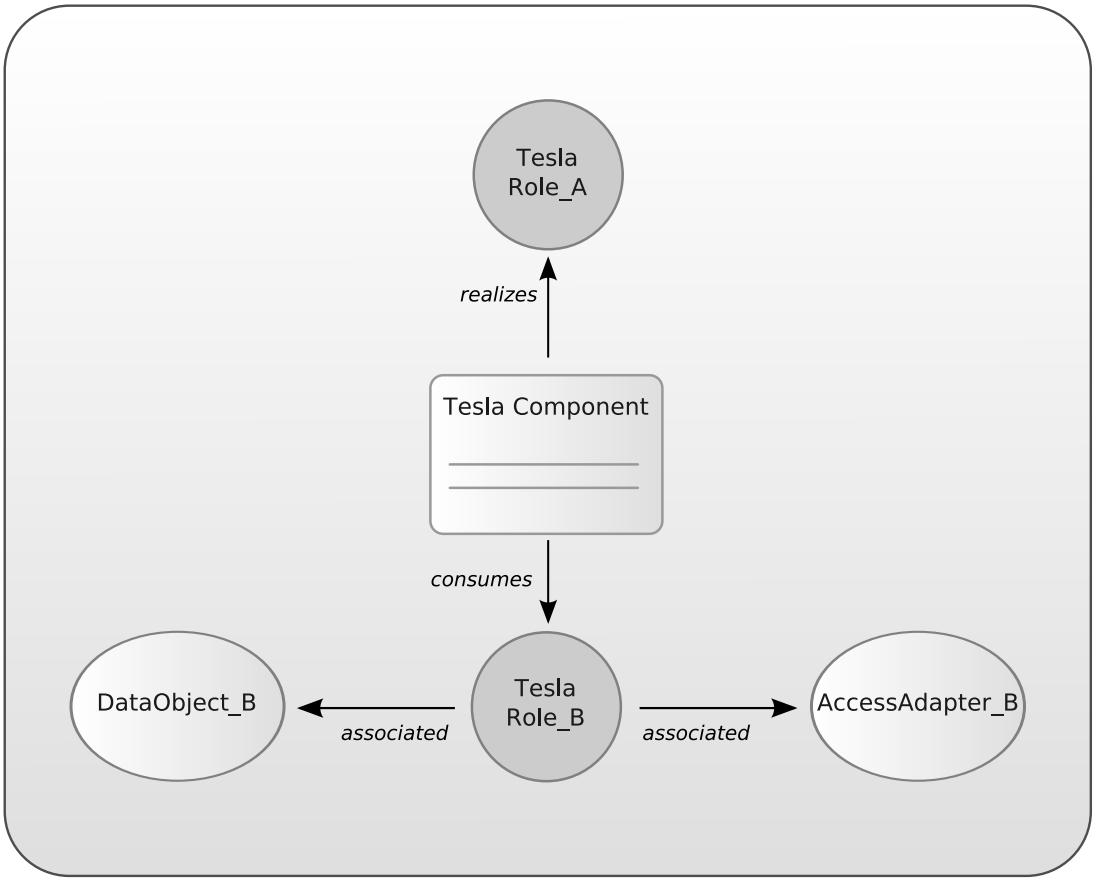


Abbildung 3.7: *Während die Output-Schnittstellen von Komponenten durch die Realisierung der Rollen festgelegt werden (Abbildung 3.6) erfolgt die Definition der Input-Schnittstelle (von der Daten konsumiert werden) innerhalb einer Komponente. Abbildung übernommen aus Hermes & Schwiebert (2010:290).*

Der Preis für die Einführung von Rollen ist eine gesteigerte Komplexität für Komponentenentwickler: Um eine Komponente in das Tesla-System einzubringen, kann man im besten Fall auf eine bestehende Rolle zurückgreifen, ansonsten muss man selbst eine solche Rolle definieren. In letzterem Fall müssen eigene DataObject- und AccessAdapter-Schnittstellen entwickelt sowie Klassen implementiert werden, welche diese Interfaces realisieren. Um diese konzeptuelle Arbeit zu erleichten und auch um die Rollen in eine übersichtliche Ordnung zu bringen (schließlich muss auch der Anwender, der Experimente realisiert, den Überblick über die zur Verfügung stehenden Werkzeuge behalten), werden die Tesla-Rollen in einem hierarchischen System abgebildet, dem Tesla-Rollensystem (vgl. Hermes & Schwiebert 2010).

Wurzel in diesem Rollensystem ist die Rolle **Annotator**, da alle Tesla-Komponenten ihre

Ergebnisse in Annotationen speichern. Hierarchisch tiefer stehende Rollen sind hinsichtlich der DataObject- und AccessAdapter-Schnittstellen mit den hierarchisch höherstehenden Rollen kompatibel, so dass die Grundfunktionalitäten dieser Schnittstellen übernommen werden können.[30] Von Annotator abgeleitet ist etwa die Basisrolle **Anchored Element Generator**. Sie inkludiert alle Rollen, die detektierte Einheiten in Texten annotieren und besitzt folgende Unterrollen:

- **Sequence Annotator**: Auszeichnung nicht überlappender Einheiten (abgeleitete Rollen sind z.B. **Tokenizer** und **Sentence Detector**)
- **Categorizer**: Auszeichnung evtl. überlappender Einheiten mit Kategorien (abgeleitete Rollen sind z.B. **POS Tagger**, **Constituent Tagger**)
- **Linker**: Setzen von Assoziationen zwischen Annotationen, z.B. von der abgeleiteten Rolle **Pronominal Reference Resolver**

Weitere, direkt von Annotator abgeleitete Rollen sind **Vector Generator** und **Clusterer**. Erstere generiert numerische Repräsentationen in Vektoren für die prozessierten Daten, zweitere erzeugt Cluster über derartige Daten. Jede dieser Rollen hat wiederum spezialisierte Unterrollen, die Vektoren eines bestimmten Datentyps (z.B. **Double Vector Generator**) oder die z.B. gewichtete Cluster erzeugen (**Weighted Clusterer**). Alle Rollen können wiederum zu spezielleren Rollen abgeleitet werden. Die für diese Basisrollen implementierten DataObjects und AccessAdapter können als Oberklassen für die Implementierungen eigener Rollen genutzt werden. Im für den Komponentenautoren günstigsten Fall existiert die benötigte Rolle mit allen assozierten und zu implementierten Elementen bereits.[31] Darüber hinaus bietet die auf Eclipse basierende Tesla IDE diverse Funktionalitäten für die Erleichterung der Komponentenerstellung.[32]

Mit dem Tesla-Rollensystem wird dem Dilemma begegnet, dass aus Sicht des Frameworks und aus Sicht der Komponente gegensätzliche Anforderungen an die Komponentenschnittstellen gestellt werden. Die vom Framework geforderten Restriktionen für die Schnittstelle werden nicht *global* (für alle Komponenten geltend), sondern *lokal* (für die Komponenten

30 In Ausnahmefällen kann die gesamte Funktionalität *entweder* des DataObjects *oder* des AccessAdapters übernommen werden. Werden beide übernommen, benötigt man keine neue Rolle.

31 Im Anhang D findet sich ein Ausschnitt der aktuellen Tesla-`roles.xml`-Datei sowie ein Verweis auf die Online-Ressource mit der vollständigen aktuellen Rollendefinition. Selbst entworfene Rollen müssen nicht in der zentralen roles.xml des Tesla-Projektes gespeichert werden, sondern können in den Komponenten selbst definiert werden. Tesla stellt sich die Rollenhierarchie aus den verteilten Ressourcen selbst zusammen.

32 Ein Tutorial zur Komponentenerstellung findet sich unter `http://tesla.spinfo.uni-koeln.de/custom_components.html`, zuletzt aufgerufen am 11.10.2011.

spezifischer Rollen geltend) gesetzt. Damit sind Komponenten einerseits in der Lage, sehr spezifische und damit aussagekräftige Ergebnisse zu erzeugen, auf der anderen Seite sind diese durch die Rollenrestriktionen für andere Komponenten zur Weiterverarbeitung auch auf eine komfortable Weise zugänglich. Für den Anwender, der Komponenten zu Experimenten zusammenstellt, sind jeweils nur die Input- und Output-Rollen der Komponente sichtbar. Dadurch genügt es, die Kompatibilität dieser Rollen zu gewährleisten, was möglich ist, ohne das vollständige Rollensystem in seiner gesamten Komplexität zu kennen (vgl. 3.2.4).

Das Tesla-Rollensystem wird zum gegenwärtigen Zeitpunkt konzeptuell weiter ausgearbeitet und ist demnach noch wandlungsfähig. Das Verfahren, mit dem die Interaktion von Komponenten auf der Tesla-Plattform realisiert werden kann, ist zwar grundsätzlich praktikabel. Gegenwärtig wird allerdings überprüft, ob die Kompatibilität von Komponenten weiter verbessert werden kann, etwa durch die Einbeziehung neuerer Entwicklungen aus dem Bereich des Semantic Web, wie die Technologien RDF[33] oder Topic Maps[34] in Kombination mit neueren reflektiven Programmiertechniken.[35]

3.2.3 Feinabstimmung: Konfiguration von Komponenten

Komponenten haben zwar exakt definierte Schnittstellen und sollten zuverlässige, rekonstruierbare Ergebnisse erzeugen. Das bedeutet aber nicht, dass Komponenten als statisch betrachtet werden können und in allen Fällen auf die exakt gleiche Weise arbeiten. Stattdessen sollten sie in einem bestimmten Maße konfigurierbar sein, um auf unterschiedliche Art eingesetzt werden zu können. Das heißt, der Verwender der Komponenten sollte von außen, ohne den Code zu ändern, Einfluss auf die Arbeitsweise der Komponente nehmen können.

Die speziellen Konfigurations-Anforderungen von Tokenizern, bei denen sowohl die Ermittlung valider Einheiten als auch deren Klassifikation von einer Konfigurationsdatei aus

33 RDF steht für *Resource Description Framework*. Es ist eine Kernkomponente des *Semantic Web* und hat zum Ziel, die Prinzipien des WWW (Verknüpfung, Offenheit, Heterogenität) von Dokumenten auf allgemeine Daten zu übertragen.

34 Das Konzept von *Topic Maps* beschreibt ein XML-basiertes Datenformat zur Formulierung von Wissensstrukturen mit dem Ziel, bessere Navigation und Suche in Internetressourcen und anderen Dokumenten zu ermöglichen sowie den Austausch von Metadaten zu verbessern.

35 Ausfühlicheres zum TRS findet sich bei Schwiebert (2011: Kapitel 4.1), wo v.a. auf die Vorteile eingegangen wird, die sich durch Nutzung des TRS für die Interaktion von Komponenten ergeben, speziell gegenüber dem rein datenorientierten Ansatz anderer linguistischer Komponentensysteme wie GATE und UIMA.

gesteuert werden konnte, wurden bereits thematisiert (3.1.4). Auch bei anderen Textprozessierungsaufgaben ist eine solche Konfigurationsschnittstelle vonnöten: In einer Reader-Konfiguration sollten etwa Ersetzungseinheiten (die Ersetzung eines langen Fraktur-ſ in normales s, Überführung von Ligaturen in Umlaute etc.) oder eine Auswahl über die zu interpretierenden Metainformationen und Annotationen, die sich in der Textressource finden, definiert werden können. Tagger könnten unterschiedliche Tagsets für die Auszeichnung anbieten, Gazetteer verschiedene und erweiterbare Abkürzungslisten. Clusterer oder Chiffrierer können eine Auswahl von Verfahren anbieten, nach denen geclustert bzw. verschlüsselt werden soll, bzw. welche Statistiken überhaupt errechnet werden sollen und ob auf Groß- und Kleinschreibung geachtet werden muss oder diese ignoriert werden kann. Diese Aufzählung lässt sich beliebig fortführen, und es ist unmittelbar einsehbar, dass es unsinnig wäre, für alle Kombinationen von Konfigurationen eigene Komponenten erstellen zu müssen. Stattdessen müssen Komponenten in einem Maße dynamisch sein, als dass sie eine Konfigurationsschnittstelle anbieten, in der bestimmte Parameter als Grundlage für die Prozessierung gesetzt werden können. Wichtig für die Nachvollziehbarkeit der Experimente mit konfigurierbaren Komponenten ist dabei stets die exakte Protokollierung der vorgenommenen Konfiguration.

Konfigurationen gehören demnach genauso in das Protokoll eines Experiments wie die Auswahl von Komponenten und Analysedaten. Tesla bietet mit der `Experiment.xml`-Datei ein Format für ein solches Protokoll an (vgl. Kapitel 3.2.4). Neben dieser Forderung von wissenschaftstheoretischer Seite bietet sich dem Anwender aber auch eine komfortable Schnittstelle, um Konfigurationen vorzunehmen, ohne den Code der Komponente ändern zu müssen. Wie in Kapitel 3.2.1 gezeigt, legt der Komponentenautor über `@Configuration` Annotationen fest, welche Schnittstellen zur Konfiguration die Komponente anbietet. Für den Nutzer der Komponente steht ein eigener Konfigurations-View mit verschiedenen eigenen Editoren zur Ansteuerung zur Verfügung. Diese sind notwendig, da auf ganz unterschiedliche Weise konfiguriert werden kann. Die Information über die Beachtung bzw. Missachtung von Groß- und Kleinschreibung etwa lässt sich durch einen Boolean-Wert ausdrücken, im zugehörigen Editor muss dann dementsprechend `true` bzw. `false` ausgewählt werden. Abkürzungen oder Gazetteer-Kategorien lassen sich am besten in Listen fassen. Abbildung 3.8 zeigt eine Reihe von unterschiedlichen Konfigurationseditoren.

Um nicht bei jeder Komponentenverwendung jede Konfigurationseinstellung manuell eingeben zu müssen, gibt es die Möglichkeit, Default-Werte zu vergeben, die immer dann gesetzt werden, wenn die entsprechende Konfigurationsschnittstelle nicht explizit bedient

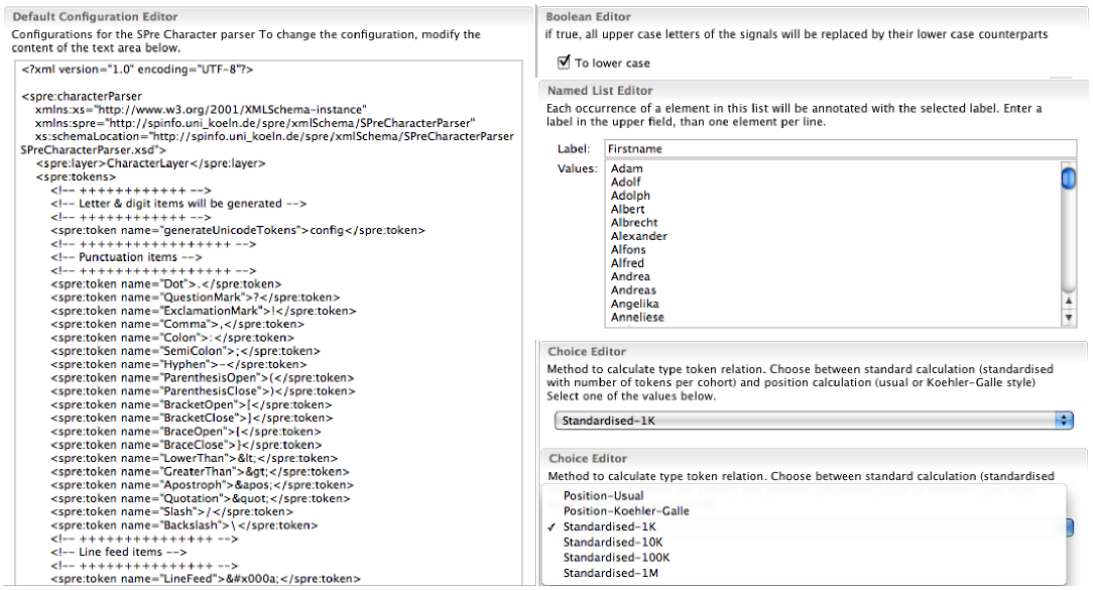


Abbildung 3.8: *Übersicht über die in Tesla bislang verfügbaren Konfigurationseditoren: Links der Default-Editor, in dem bspw. auf XML basierende Konfigurationen editiert werden können. Rechts spezialisierte Editoren (Boolean, Listen, Auswahl).*

wurde. Darüber hinaus wurden in Tesla mit `Templates` und `TemplateSets` Möglichkeiten geschaffen, Auswahllisten von Konfigurationsparametern und Kombinationen einzelner Werte als Set abzulegen. Beispielhaft verdeutlicht werden kann dies anhand der `CryptographicSubstitutionComponent` (siehe auch Anhang C): Die Komponente bietet eine Reihe von Verfahren an, mit denen die Tokens des Inputs verschlüsselt werden können. Für jedes dieser Verfahren wurde ein Template angelegt, das aus einem XML-Element besteht. Um von Tesla aufgefunden zu werden, muss sich dieses Element innerhalb eines `templates`-Element in einer Datei `templates.xml` befinden, die im Ordner `META-INF` der betreffenden Tesla-Komponente liegen muss. Das Template-Element enthält

1. Den Namen des Templates im Attribut “name”.
2. Die Kurzbeschreibung für das Template im Element “description”.
3. Den Wert des Templates im Element “value” – Dieser wird von der Komponente interpretiert und so der Code für das entsprechende Verfahren ausgewählt.
4. Die Kategorie des Templates im Attribut “category” – Diese muss der Kategorie des Java-Elements entsprechen, das die `@Configuration`-Annotation trägt.
5. Eine eindeutige ID innerhalb der `templates.xml`-Datei zur Verwendung innerhalb von *TemplateSets* (s.u.).

Ein solches `Template` für das Caesar-Substitutionsverfahren (vgl. 5.1) aus der `Cryptographic`

`SubstitutionComponent` findet sich in Listing 3.6.

```
1 <template id="cae" name="CaesarCipherGenerator"
2    category="Substitution Method">
3    <description>Method of substitution</description>
4    <value>CAESAR</value>
5 </template>
```

Listing 3.6: *Beispiel für ein Konfigurations-Template: Caesar-Chiffre.*

Das Template lässt sich mit mehreren anderen Templates kombinieren, die für das Caesar-Verfahren sinnvolle Werte enthalten. Da das Verfahren symmetrisch ist (es kann sowohl zur Ver- als auch zur Entschlüsselung genutzt werden), fehlt z.B. die Angabe, in welche Richtung das Verfahren angewendet werden soll: Soll damit ein Klartext verschlüsselt werden (mit einem Encipherer) oder soll das Verfahren auf einen Geheimtext zur Entschlüsselung angewendet werden (mit einem Decipherer), mit welchem Schlüssel soll das Verfahren angewendet werden, welches Verschlüsselungsalphabet liegt zugrunde, sollen eventuell Einheiten aus dem Klartext ersetzt werden (z.B. Umlaute), sollen Spatien gelöscht werden, um die Entschlüsselung zu erschweren? Derartige Kombinationen von Templates werden in TemplateSets zusammengefasst, die aus Referenzen auf die einzelnen Templates bestehen. Listing 3.7 zeigt ein solches TemplateSet für einen Caesar-Chiffren-Generator mit den zugehörigen Templates.

```
1 <set id="3"> <!-- Reihenfolge fuer Darstellung im Tesla-Editor -->
2   <name>Ceasar Cipher Generator</name>
3   <description>Generates ciphers based on Caesars method</description>
4   <template id="cae"/>           <!-- Template fuer Verfahren -->
5   <template id="enc"/>           <!-- Anwendungsrichtung -->
6   <template id="int_key"/>       <!-- Default-Key -->
7   <template id="rep_german"/>    <!-- ErsetzungsEH, z.B. Umlaute -->
8   <template id="del_ws"/>        <!-- Template zum Leerzeichen loeschen -->
9 </set>
```

Listing 3.7: *Beispiel für ein TemplateSet: Auswahl passender Werte für die Caesar-Chiffre. Die zugehörigen Templates finden sich in Anhang D.*

Die Sets werden innerhalb des Clients im Experiment-Configuration-View dargestellt, wenn man die zu konfigurierende Komponente auswählt. Nach der Auswahl eines Sets (oder einzelner Templates) können die Werte noch innerhalb der entsprechenden Editoren angepasst werden.

3.2.4 Instanzen wissenschaftlicher Arbeit: Experimente

Wenn Komponenten einzeln konfiguriert und miteinander verknüpft werden, entsteht dadurch ein Versuchsaufbau. Können mit diesem Versuchsaufbau Textsammlungen prozessiert werden, so kann man dies *textprozessierendes Experiment* nennen. Wie es in einem naturwissenschaftlichen Labor durch ein Laborbuch vorgesehen ist, werden Experimente auf das genaueste protokolliert, um die erzielten Ergebnisse reproduzierbar zu halten. Im folgenden wird zunächst darauf eingegangen, wie sich Experimente mit textprozessierenden Komponenten in Tesla realisieren lassen. Daran anschließend wird gezeigt, wie ein solches Experiment persistiert wird, damit es wiederholt oder auch weitergegeben werden kann.

Für die einfache Erstellung von Experimenten steht in Tesla ein graphischer Editor zur Verfügung. Textselektionen und Komponenten können einfach per *drag and drop* aus dem *CorpusManager-* bzw. dem *Components-View* zu einem Experiment zusammengefügt werden. Um Ein- und Ausgangsschnittstellen der Komponenten miteinander verknüpfen zu können, müssen die Rollen dieser Komponenten kompatibel sein. Ist innerhalb eines Experiments eindeutig entscheidbar, welcher Output welche Input-Schnittstelle bedient (das ist der Fall, wenn nicht mehrere Komponenten die gleiche Rolle erfüllen), übernimmt der Editor die Verknüpfungen der Komponenten automatisch, ansonsten müssen sie vom Nutzer selbst gesetzt werden. In Abbildung 3.9 ist ein Experiment dargestellt, das eine vorgegebene Textselektion tokenisiert, die Tokens dann lemmatisiert und gegebenenfalls mit dem Gazetteer kategorisiert. Zur Konfiguration der einzelnen Komponenten, etwa zur Auswahl oder Erweiterung der Gazetteer-Listen, muss im Configuration-View der betreffende Reiter ausgewählt werden.

Ist das Experiment fertig konfiguriert, kann es über die entsprechende Schaltfläche (Run-Button, Mitte der Menüzeile) gestartet werden.[36] Anhand des *Experiment-State-View* (unterer Bildrand) lassen sich die Fortschritte verfolgen. Beendete Experimente erscheinen im Experiment View; ihre Ergebnisse können von dort aus über "Analyze..." betrachtet werden. Dabei wird zunächst ein *Experiment Report* generiert, der die wichtigsten Parameter des Experiments auflistet (Ausführender, Zeitpunkt und Status der Ausführung, Anzahl der geschriebenen Annotationen einzelner Komponenten) und in dem einzelne Komponenten evtl. kompakt darzustellende Werte in einer übersichtlichen Darstellung ablegen können. Aus dem Experiment Report heraus kann dann festgelegt werden, über

36 Ausführliche Tutorials zu Tesla finden sich unter `http://tesla.spinfo.uni-koeln.de/tutorials.html`, zuletzt aufgerufen am 11.10.2011.

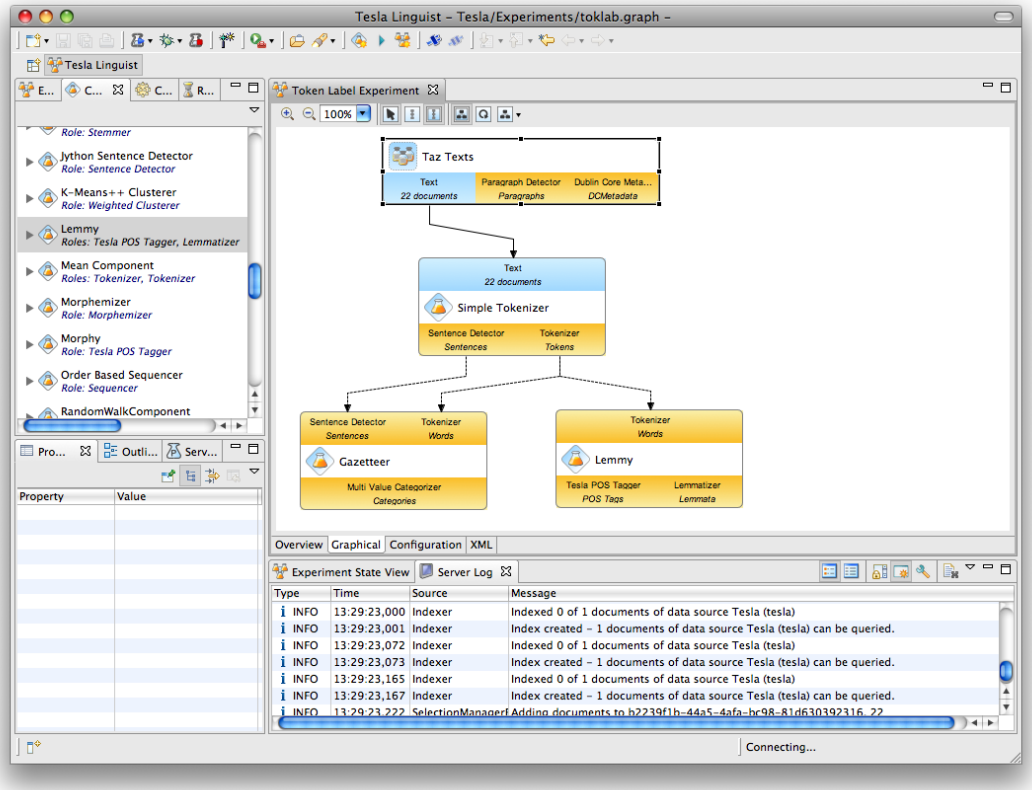


Abbildung 3.9: *Client-Ansicht von Tesla mit graphischem Experiment-Editor. Text-Sammlungen und Komponenten können per* drag and drop *aus den Views links (CorpusManager und TeslaComponents) zum Experiment hinzugefügt werden.*

welche Texte welche Ergebnisse welcher Komponente in ein generisches XML-Format exportiert werden sollen. Dieses XML-Format ist Basis für die weitere Verarbeitung oder Veröffentlichung, zu der die Daten über eine Reihe von XSL-Operationen transformiert werden können. Dies könnte z.B. die Weitergabe (z.B. in einem TEI-konformem Format) oder die Visualisierung der Ergebnisse mit unterschiedlichen Tools sein. Bisher integrierte Visualisierer erzeugen farblich hinterlegten Text, Klammerstrukturen, Tabellen (auch als csv-Export) und eine Cloud-View (dabei werden Wörter in verschiedenen Größen – gewichtet nach ihrer Frequenz – dargestellt).[37] Tesla ist für die Einbindung weiterer Visualisierer (z.B. eines Datenplotters) offen.

Das gesamte Experiment, d.h. Referenzen auf die Texte, sämtliche beteiligten Komponenten inkl. Versionsnummer und Konfiguration, die gesamte Interaktion der Komponenten

37 Unter `http://tesla.spinfo.uni-koeln.de/experiment_evaluation_advanced.html` (zuletzt aufgerufen am 11.10.2011) findet sich eine Darstellung der unterschiedlichen Visualisierer.

(was ist wie miteinander verknüpft?) – wird in einer XML-Datei persistiert und kann damit einfach distribuiert werden. Ziel dieser Persistierung ist es, den experimentellen Aufbau in eine 1:1-Beziehung mit den resultierenden Ergebnissen zu setzen. Durch Weitergabe des experimentellen Aufbaus lassen sich die gewonnenen Ergebnisse auf unkomplizierte und effektive Art distribuieren. Das nächste Kapitel erläutert, welche zusätzlichen Features in Tesla implementiert wurden, um diese Weitergabe zu vereinfachen.

3.3 Ein Labor zur kooperativen Wissenschaft

Für die Entwicklung eines virtuellen Labors zur Textprozessierung ist es ein zentrales Anliegen, zu gewährleisten, dass Experimente unkompliziert zwischen Wissenschaftlern und Laboratorien ausgetauscht werden können. In Tesla sind es im Wesentlichen vier Features, welche diese Kooperationsfähigkeit unterstützen. 3.3.1 zeigt, wie Texte in Tesla über eine ID referenziert werden, die in jeder Tesla-Instanz für gleiche Texte stets dieselbe ist. Inwieweit abweichende Komponentenversionen in Tesla unterschieden werden können, zeigt 3.3.2. 3.3.3 geht darauf ein, wie vermieden werden kann, dass (Teil-)Experimente innerhalb einer Tesla-Instanz immer wieder neu ausgeführt werden. Damit werden Prozessierungszeit und Speicherplatz für redundante Berechnungen und Daten minimiert. 3.3.4 schließlich geht kurz auf die mehrschichtige Architektur von Tesla ein und führt dabei die Vorteile auf, die sich dadurch für das kooperative Arbeiten ergeben.

3.3.1 Wiedererkennung von Dokumenten

Die Rohdaten, die in Tesla-Experimenten prozessiert werden, sind über ihre Referenzierung immer Teil eines Experiments. Wird ein Experiment also in Form der `experiment.xml`-Datei für andere Wissenschaftler veröffentlicht, so muss hinsichtlich dieser Rohdaten zweierlei gewährleistet werden:

1. Da die Rohdaten nicht vollständig in die `experiment.xml`-Datei aufgenommen werden können, müssen sie anderweitig zur Verfügung gestellt werden.
2. Die Referenz allerdings muss die gleiche bleiben, d.h. der Text, der beim Veröffentlicher des Experiments mit *a123* referenziert wird, muss auch in allen anderen Tesla-Instanzen mit *a123* referenziert werden.

Zu diesem Zweck werden TeslaDocuments mit einer ID versehen, die dem MD5-Wert[38] ihrer Daten entspricht. Mit Berechnung des MD5-Wertes ist gewährleistet, dass ein Text, der aus der gleichen Zeichenfolge besteht, immer dieselbe ID erhält.[39] Damit ist es möglich, Experimente auch unabhängig von der verwendeten Datenbasis weiterzugeben. Legt man seinen Analysen bspw. das BNC zugrunde, so sind die Experimente in jeder Tesla-Instanz reproduzierbar, die Zugriff auf das BNC hat. Alternativ können die experimentell verwendeten Texte separat weitergegeben werden (etwa in einem gepackten Format). Der Empfänger kann diese Texte einfach in seiner Tesla-Installation registrieren (vgl. 3.1.1) und die ebenfalls importierten Experimente ausführen.

3.3.2 Versionierung von Komponenten

Die Forderung nach exakter weltweiter Reproduzierbarkeit steht in Konflikt mit der Dynamizität von Software. Theoretisch müsste der Code jeder Komponente, die an einem Experiment beteiligt ist, inklusive aller Bibliotheken, auf die die Komponente zugreift, eingefroren werden, damit das Experiment exakt reproduzierbar bleibt. Bisher wurde auf einen solchen Mechanismus in Tesla verzichtet, da er die Entwicklungsphase von Komponenten zu sehr beeinträchtigen und zu einer unübersichtlich großen Anzahl von Komponentenversionen führen würde. In folgenden Tesla-Versionen soll die automatische Versionierung und die Verwaltung von Komponentenversionen implementiert werden. Bis dahin lassen sich Komponenten zumindest manuell versionieren, indem der Autor Zugriff auf das Version-Attribut der Komponente hat. Der Autor bzw. Anwender muss also selbst dafür sorgen, dass veröffentlichte Komponenten in dieser veröffentlichten Version verfügbar bleiben und zukünfige Änderungen unter einer neuen Versionsnummer eingebaut werden. Das Versions-Attribut ist ein `String`, so dass man hinsichlich der Bezeichner für Versionen frei ist, diese sich also z.B. personalisieren lassen, beispielsweise über Namenskürzel mit Versionsnummer (`version = ‘‘jhermes3.0’’`).

38 Der Message-Digest Algorithm 5 (MD5) ist eine weit verbreitete kryptographische Hashfunktion, die aus einer beliebigen Nachricht einen 128-Bit-Hashwert erzeugt. Entwickelt wurde der Algorithmus von Rivest (1992).

39 Umgekehrt ist es bereits häufiger gelungen, Schlüsselkollisionen zu generieren. Allerdings ist dies nur mit relativ hohem Aufwand möglich und hat eher Auswirkungen auf die Anwendungen von MD5-Werten in der Informationssicherheit (vgl. Schmidt). Dass tatsächlich zwei unterschiedliche Texte eines Korpus den gleichen MD5-Wert zugewiesen bekommen, dürfte so gut wie ausgeschlossen sein.

3.3.3 Wiederverwendung von Ergebnissen

Bei der experimentellen Arbeit kommt es des öfteren vor, dass man Experimente wiederholen muss, weil man z.B. eine andere Clustermethode verwenden will. Man stelle sich vor, man habe dafür ein umfangreiches Korpus präprozessiert, evtl. PoS-Tags vergeben, Kookkurrenzvektoren für Nomen erstellt, um diese mittels eines bestimmten Cluster-Verfahrens (z.B. *k-Means*) zu fünf Ähnlichkeitsclustern zusammenzufassen. Nun soll genau das gleiche Ausgangsmaterial auf zehn Cluster aufgeteilt werden. Präprozessierungs-, Tagger- und Vektorerzeugungs-Komponente noch einmal laufen zu lassen würde unnötige Prozessierungszeit und vor allem unnötigen Speicherplatz im Cache bzw. der Datenbank beanspruchen, da diese Komponenten exakt auf dem gleichen Datenmaterial mit exakt der gleichen Konfiguration prozessiert haben. Die von den Komponenten im ersten Durchlauf erzeugten Ergebnisse sind also wieder exakt die Ergebnisse des zweiten Durchlaufs (jedenfalls soweit die Komponenten – wie im vorliegenden Fall – deterministisch arbeiten). Deshalb könnten erstere für die Cluster-Komponente (die ja abweichend konfiguriert wurde und daher wirklich neu prozessieren muss) zugänglich und so die wiederholte Prozessierung überflüssig gemacht werden.

Konkret wurde der Mechanismus zur Wiederverwendung von Ergebnissen so implementiert, dass Komponenten in Experimenten durch Komponenten anderer Experimente (die sich im Laborbuch, also der Datenbank, befinden) ausgetauscht werden können. Damit ein solcher Austausch auch wirklich nur dann erfolgt, wenn exakt die gleiche Komponente schon einmal gelaufen ist, müssen folgende Bedingungen erfüllt sein:

1. Die Input-Daten und die Konfiguration der Komponente müssen dieselben sein.
2. Die Komponente muss ihre Prozessierung ohne Fehler beendet haben.
3. Die Komponente muss dieselbe Versionsnummer aufweisen.
4. Sie muss das Konfigurations-Attribut "ReusableResults" auf true gesetzt haben.[40]

Während für die eindeutige Bestimmung der Konfiguration die Berechnung des MD5-Wertes ausreicht, muss die Berechnung der Input-Schnittsttelle(n) iterativ erfolgen, gemäß dem Graph der Komponentenzusammenstellung. Verdeutlicht wird die Berechnung anhand des folgenden Experiments (Abbildung 3.10): Zunächst wird überprüft, ob der

40 Der Wert *true* darf deshalb auch nur an deterministisch arbeitende Komponenten vergeben werden (Komponenten, in denen es keine zufälligen Verfahrensänderungen gibt bzw. bei denen zufällige Ereignisse nicht reproduzierbar sind). Während der Entwicklung von Komponenten, bei der man daran interessiert ist, zu Testzwecken mehrfach zu prozessieren, kann man den Wert auf *false* setzen, ebenso wenn man der Meinung ist, die Prozessierung dauere kürzer als ein Lookup in der Datenbank oder im Cache.

Reader dieselbe Konfiguration hat und auf denselben Texten operiert (durch die ID der Texte können diese ja eindeutig referenziert werden, vgl. 3.3.1). Der berechnete Kennwert des Readers geht ein in die Berechnung des Kennwertes der nachfolgenden Komponente, in diesem Fall die des Tokenizers. Konkret wird hier der kombinierte MD5-Wert für den Input und die Konfiguration des Tokenizers errechnet. Das Verfahren wiederholt sich für den Vektorersteller und schließlich den Clusterer.

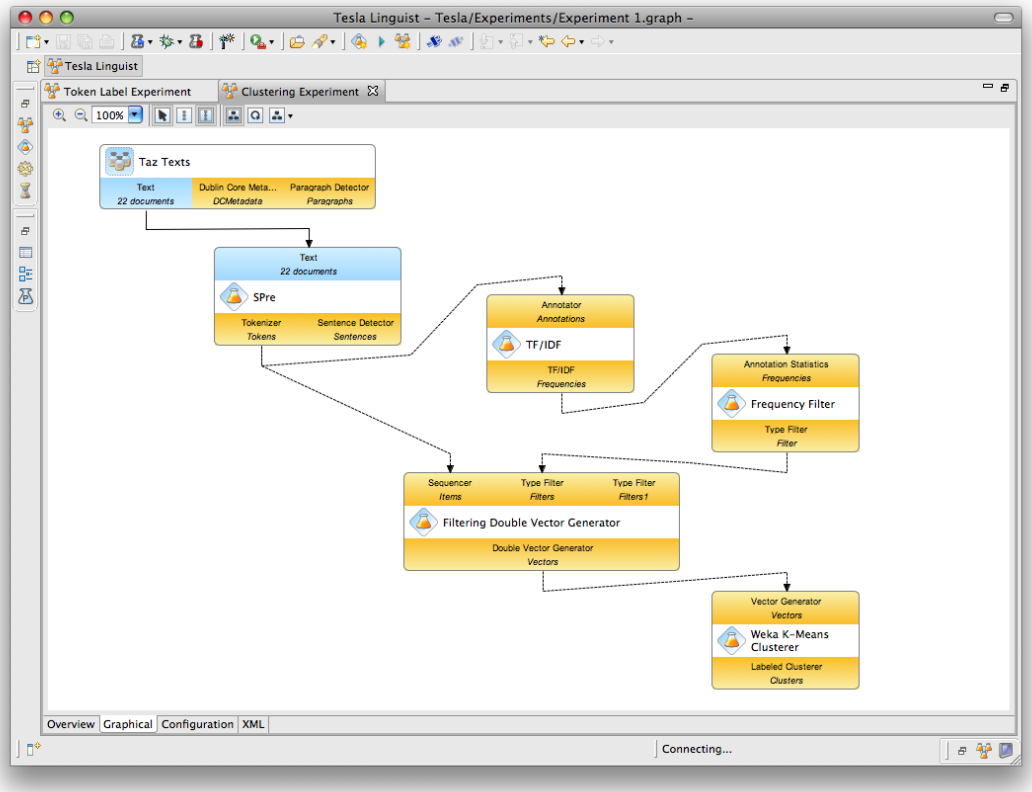


Abbildung 3.10: *Experiment zur Clusterung von Daten im Workflow-Editor.*

Bei der ersten Ausführung des Experiments müssen noch alle Komponenten prozessieren. Wird im Anschluss daran aber lediglich der Clusterer modifiziert (also z.B. die Anzahl der Cluster verändert oder das k-means-Verfahren durch einen anderen Algorithmus – z.B. das *EM*-Verfahren – ausgetauscht), ändert sich lediglich die Konfiguration der letzten Komponente im Graphen und damit auch nur deren MD5-Kennwert. Bei der wiederholten Ausführung registriert Tesla, dass alle Komponenten bis auf den Clusterer bereits einmal prozessiert haben und führt sie deswegen nicht noch einmal aus. Stattdessen erhält der Clusterer Zugriff auf die Vektoren des im ersten Durchgang gelaufenen Vektor-

Erzeugers. Wenn man die Konfiguration des Präprozessors modifiziert, dann müssen alle Komponenten – bis auf den Reader – ausgeführt werden. Dies ist dadurch gewährleistet, dass sie ab dem Präprozessor abweichende MD5-Kennwerte erhalten haben. Bekommt eine Komponente Input von mehreren vorher gelaufenen, so müssen alle Kennwerte der Input-Komponenten in die neue Berechnung einfließen.

3.3.4 Client-Server-Architektur

Arbeitsplatzrechner sind meist darauf optimiert, dem Benutzer eine Schnittstelle zu vielfältigen Anwendungsprogrammen zu bieten. Der Benutzer möchte meist mehrere Anwendungen im Wechsel bedienen (Mail-Client, Browser, Textverarbeitung, Programmier-IDE etc.) und ist deshalb darauf angewiesen, dass die begrenzten Ressourcen des Arbeitsplatzrechners nicht durch extrem rechenintensive Anwendungen blockiert sind. Daher bietet es sich an, Prozesse, die unter Umständen extreme Rechenkapazitäten oder die Verwaltung enormer Datenmengen erfordern, auf dafür spezialisierte Maschinen auszulagern. Die Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerks zu verteilen, wird im *Client-Server-Modell* beschrieben. Server sind dabei Programme, die bestimmte Dienste (Services) anbieten, die von Clients genutzt werden. Die Kommunikation zwischen Client und Server wird über ein Protokoll abgehandelt, das spezifisch für die jeweiligen Dienste ist und festlegt, Informationen welcher Art ausgetauscht werden können. Server sind ständig in Bereitschaft, verhalten sich aber passiv, bis ein Client aktiv einen Dienst anfordert. Prinzipiell kann der Server auf mehrere physische Maschinen aufgeteilt sein und mit einer Reihe von Clients kommunizieren. Das Client-Server-Modell kann in unterschiedlich vielen Schichten realisiert werden. Bei einer Zwei-Schichten-Architektur wird der Server fast ausschließlich als Datencontainer genutzt, die Anwendungslogik läuft größtenteils auf dem (Fat-)Client, auf dem Server findet sich lediglich ein Datenbankprogramm. Bei einer Drei-Schichten-Architektur findet sich neben der Datenhaltungsschicht auch eine Applikationsschicht auf dem Server, in welcher die Daten manipuliert werden. Der (Thin-)Client ist so vom Prozessierungsaufwand befreit und enthält lediglich die Darstellungsschicht.

Auf eine solche Drei-Schichten-Architektur hin wurde das System Tesla angelegt. Der Tesla-Client prüft im Hintergrund, ob ein Tesla-Server erreichbar ist. Sobald ein solcher gefunden ist, versucht der Client sich zu verbinden. In diesem Zuge verifiziert der Server den Client anhand seiner Benutzerdatenbank. Nun können vom Client aus Korpora auf dem Server angelegt oder erweitert und Selektionen über die Korpustexte geschaffen werden. Im Client werden auch Experimente zusammengestellt und konfiguriert. Nach

Abschluss der Erstellung können Experimente gestartet werden, indem sie zum Server geschickt werden, der diese ausführt und die Ergebnisse in die entsprechenden Datenbanken persistiert. Während der Prozessierung kann sich der Client durchgehend über den aktuellen Stand der Ausführung beim Server informieren. Nachdem alle im Experiment definierten Prozesse abgeschlossen sind, kann man vom Client aus die Ergebnisse anfordern, wie dies in 3.2.4 ausgeführt wurde. Die beschriebene Arbeitsteilung zwischen Client und Server wird schematisch in Abbildung 3.11 dargestellt.

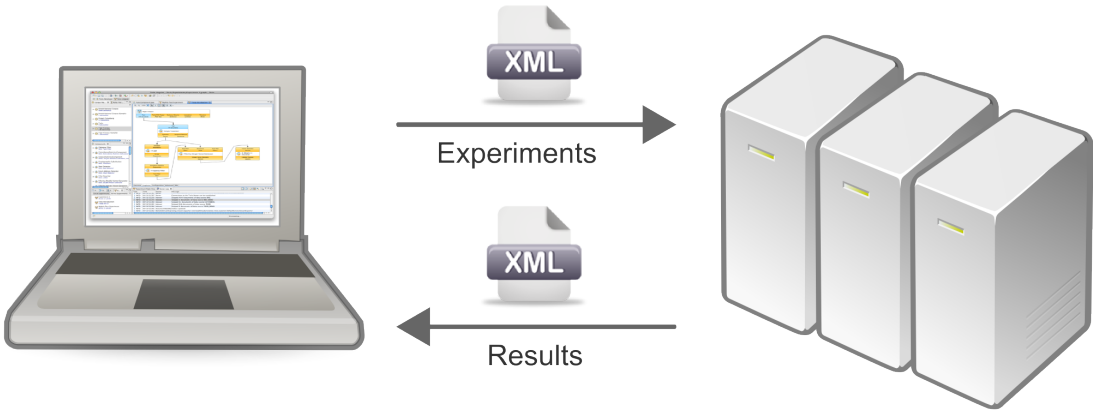


Abbildung 3.11: *Austausch von Client und Server in Tesla. Die auf dem Client zusammengestellten Experimente werden auf dem Server ausgeführt, der wiederum dem Client die Ergebnisse zugänglich macht.*

Technisch gesehen basiert die Applikationsschicht des Tesla-Servers auf dem *Spring-Framework*, das es ermöglicht, die zentralen Konzepte des für Tesla entworfenen Komponentenframeworks umzusetzen, zugleich aber auch in jeder beliebigen JavaSE-Umgebung ausgeführt werden kann. Aufgrund der sehr unterschiedlichen Anforderungen, welche Komponenten im Zuge ihrer Ergebnispersistierung an das Persistenzframework stellen, wurde kein einzelner Mechanismus gefunden, der alle diese Ansprüche befriedigt hätte. Durch die Adapter-Schnittstellen von Komponenten können stattdessen mehrere verschiedene Persistenzmechanismen angesprochen werden, die sehr unterschiedliche Arten der Datenspeicherung verfolgen. Zwingend ist lediglich eine lauffähige relationale Datenbank – zur Zeit werden Hypersonic (`http://hsqldb.org/`) und postgreSQL (`http://www.postgresql.org/`, beides zuletzt aufgerufen am 11.10.2011) unterstützt – um die für Tesla zentralen Entitäten (Komponenten, Rollen, Templates usw.) zu persistieren. Ergebnisse der Komponenten können ebenfalls in diese RDB geschrieben werden (Hibernate), alternativ können auch eine Objektdatenbank (DB4O) oder eine eigenentwickelte dateibasierte DB (TunguskaDB, dokumentiert bei Schwiebert 2011: Kapitel 4.2) zur Per-

sistierung der Ergebnisse genutzt werden.

Dadurch, dass der Tesla-Client durch eine Reihe Plugins als Extension der Plattform Eclipse realisiert ist, bietet sich vor allem Entwicklern von Komponenten (in der *Developer-Perspektive*, s.o.) eine Reihe von Vorteilen, weil relativ einfach auf die reichhaltige IDE-Unterstützung von Eclipse zugegriffen werden kann. Dazu gehören diverse Plugins von (Fremd-)Entwicklern, z.B. zur Versionierung, Kommunikation, Kollaboration und Visualisierung. Die Tesla-IDE selbst bietet zusätzliche Hilfestellungen (sogenannte *Wizards*) zur Erstellung von eigenen Komponenten, Rollen, Adaptern und DataObjects an. Auch aus Perspektive des Anwenders (*Linguist-Perspective*, s.o.) ist die Nutzung eines weit verbreiteten Tools wie Eclipse von Vorteil, da ein bekanntes *look and feel* die Hürden für die Benutzung einer Software absenkt.

Ein noch unerwähnter Vorteil der mehrschichtigen Architektur ist – neben der Entlastung der Client-Ressourcen – die Einrichtung einer Zentrale für den Austausch von Versuchsaufbauten und Ergebnissen. Ein Tesla-Server lässt sich so z.B. als ein zentrales virtuelles Projektlabor nutzen, in dem die einzelen Partizipanten des Projekts von verschiedenen Clients aus miteinander Experimente aufsetzen und die Resulate auswerten können. Im größeren Rahmen gedacht lassen sich die Experimente auch für eine größere Community freigeben, die von den Ergebnissen dann unmittelbar profitieren kann.

3.4 Zusammenfassung und Überleitung

In diesem Kapitel wurde das System Tesla vorgestellt, das als Arbeitsplatz für die empirisch-experimentelle Textanalyse konzipiert wurde. Die Plattform wurde offen konstruiert, um sie einem möglichst weiten Nutzerkreis zugänglich zu machen. So besteht Tesla nicht auf einem bestimmten Datenformat, vielmehr sind prinzipiell alle Daten, die sich durch Sequenzen diskreter Einheiten darstellen lassen, innerhalb des Systems analysierbar. Dies wird gewähleistet durch das *Reader-Konzept*, mit dem in dieser Weise repräsentierte Daten auf die jeweils erforderliche Weise durch Reader interpretiert werden können. Tesla bietet ein *Komponenten-Konzept*, so dass die Verarbeitung der Daten innerhalb von wiederverwendbaren, miteinander kombinierbaren, in sich abgeschlossenen Einheiten stattfinden kann. Die Schnittstellen der Komponenten sind dabei relativ frei gehalten, so dass sie aussagekräftige Ergebnisse produzieren können. Durch ein *Rollen-Konzept*, das für lokale Beschränkungen von Komponenten-Schnittstellen sorgt, wird ermöglicht, dass diese trotz dieser freien globalen Schnittstelle wiederverwendet werden können. Das Rollen-

Konzept gewährleistet in Verbindung mit dem *Konfigurations-Konzept* darüber hinaus, dass Anwender die in Komponenten gekapselten Werkzeuge nutzen und auf ihre Bedürfnisse anpassen können, ohne ihre konkrete Implementierung zu kennen bzw. Änderungen im Quellcode durchführen zu müssen.

Weil Tesla eine Plattform bietet, die von Wissenschaftlern aus verschiedenen Bereichen genutzt werden kann und weil es eine Komponentenarchitektur bereithält, welche die Anpassung von Werkzeugen auf unterschiedliche Anforderungsszenarien ermöglicht, unterstützt das System auch den Transfer von Methoden zwischen unterschiedlichen Wissenschaftsbereichen. Über diesen Methodentransfer hinaus offeriert Tesla die Möglichkeit, experimentelle Versuchsaufbauten in einer Form weiterzugeben, die eine Reproduktion gewonnener experimenteller Ergebnisse ermöglicht und unterstützt auf diese Weise einen nachhaltigen Wissenschaftbetrieb.

Die Arbeitsweise von Tesla kann am besten demonstriert werden, indem man einen konkreten Anwendungsfall mit Hilfe des Systems bearbeitet. Hierfür bietet sich ein Anwendungsbereich an, bei dessen Bearbeitung die Vielseitigkeit des Systems dargestellt werden kann, im Zuge dessen eine Reihe leicht verständlicher experimenteller Abläufe durchgeführt werden, die vorzugsweise auf Methoden beruhen, die innerhalb unterschiedlicher wissenschaftlicher Disziplinen entwickelt wurden. Darüber hinaus ist es wünschenswert, wenn es bereits eine Community gibt, welche diesen Anwendungsfall bearbeitet und die von den – hier vorrangig als Demonstration des Tesla-Systems durchgeführten – Experimenten und Ergebnissen profitieren könnte, so dass Tesla eventuell auch Teil der auf dieser Community basierten Forschung werden kann. Ein Anwendungsbereich, der diese Konditionen erfüllt, wird im nächsten Teil der Arbeit (4) eingeführt und im übernächsten (5) auf eine theoretische Basis gestellt, bevor Tesla für die tatsächliche experimentelle Anwendung in diesem Bereich genutzt wird (6).

But no one yet knows, and the book lies quietly inside the slipcase in the blackness of Kraus' vault, possibly a time bomb in the history of science, awaiting the man who can interpret what is still the most mysterious manuscript in the world.

(David Kahn: The Codebreakers)

Kapitel 4

Exemplarischer Anwendungsfall: Voynich-Manuskript

Grundlage für die Demonstration der Einsatzmöglichkeiten des Systems Tesla ist der Text eines Manuskripts, das unter der Signatur *MS 408* an der *Beinecke Rare Book and Manuscript Library* geführt wird. Die Beinecke-Bibliothek gehört zur Universität Yale, der das Manuskript von einem Buchhändler überlassen wurde. Die im Katalog zu findende Beschreibung liest sich in Auszügen folgendermaßen:

> "**MS 408 – Central Europe [?], s. XVex-XVI [?] – Cipher Manuscript.**
>
> *This manuscript is not available for consultation without advance permission from the curator.*
>
> Scientific or magical text in an unidentified language, in cipher, apparently based on Roman minuscule characters; the text is believed by some scholars to be the work of Roger Bacon since the themes of the illustrations seem to represent topics known to have interested Bacon (...) Written in Central Europe [?] at the end of the 15th or during the 16th [?] century; the origin and date of the manuscript are still being debated as vigorously as its puzzling drawings and undeciphered text."

Das Manuskript enthält einen Text unbekannten Alters, unbekannter Herkunft und unbekannten Inhalts, möglicherweise ist er mit einem unbekannten Verfahren verschlüsselt. Das Anwendungsgebiet fällt damit in den Bereich der Kryptologie, genauer: In den der Kryptoanalyse. Das Manuskript wird nach seinem Entdecker Wilfrid Voynich das *Voynich Manuskript* (VM) genannt. Es wurde für diese Arbeit als Anwendungsszenario ausgewählt, weil es geradezu einen Paradefall für communitybasierte Textprozessierung darstellt: Im Laufe der Jahre seit seiner Entdeckung haben sich eine ganze Reihe Forscher der verschiedensten Fachrichtungen daran versucht, diesem Text seine Geheimnisse zu entringen. Dabei kamen sehr verschiedene Methoden sehr unterschiedlicher Forschungsrichtungen zum Einsatz. In Tesla können die vielfältigen Ansätze in einer homogenen Umgebung zusam-

mengetragen und die Ergebnisse evaluiert sowie gegebenenfalls miteinander kombiniert werden. Wie im Laufe dieses Kapitels gezeigt wird, gab es in früheren Arbeiten zum VM gewisse Ungereimtheiten und das Manuskript zog eine Reihe von Scharlatanen an. Dies hatte zur Folge, dass sich fast niemand mehr wissenschaftlich mit ihm auseinandersetzen wollte. Gerade für einen solchen Untersuchungsgegenstand kann ein System, das die gewonnenen Erkenntnisse nachvollziehbar dokumentiert und zur Verfügung stellt, große Dienste leisten.

Diese Arbeit ist nicht so vermessen, dass sie den Anspruch hätte, die bisherigen Forschungen in ihrer Gesamtheit darzustellen, oder gar den, das Manuskript tatsächlich zu entschlüsseln. Stattdessen soll hier im Vordergrund stehen, wie Tesla als virtuelles Labor dabei helfen kann, Ergebnisse von Forschungen zusammenzuführen, um eine Plattform zu bieten, auf der weitere Analysen aufbauen können. Dazu werden im Folgenden das VM als Untersuchungsgegenstand sowie die Forschungen, die zu diesem Schriftstück durchgeführt wurden, in kompakter Form dargestellt. Zunächst werden die bisher angestellten Theorien zum Ursprung und der Geschichte des VM aufgeführt (4.1). Daran anschließend werden die wichtigsten Eigenschaften des VM erörtert (4.2), dann die Theorien über seinen Inhalt gruppiert (4.3). Zum Schluss, als Überleitung zum theoretischen Kryptologie-Kapitel 5, werden noch einmal die Eigenschaften des VM zusammengefasst, die ein ihm zugrundeliegendes Verschlüsselungsverfahren abbilden muss (4.4).

4.1 Zur Herkunft des Voynich-Manuskripts

Die Geschichte des Manuskripts lässt sich in drei Phasen einteilen, die hier jeweils in einem Unterkapitel behandelt werden. Die Reihenfolge ist dabei umgekehrt chronologisch, da die Unsicherheit über die Geschichte des VM zunimmt, je weiter man in der Zeit zurückreist. Gegenwärtig ist nur die Geschichte seit der Entdeckung des VM durch Voynich ab 1912 zweifelsfrei zu klären (4.1.1), gewisse Indizien sprechen darüber hinaus für die Tatsache, dass es bereits im 17. Jahrhundert in Mitteleuropa vorlag (4.1.2). Im Jahr 2009 wurde das VM zum ersten Mal mit der Radiokarbon-Methode datiert. Wo vorher viele Forscher seine Entstehung im ausgehenden 15. bzw. Anfang des 16. Jahrhunderts vermuteten, die ursprüngliche Theorie sogar annahm, es sei im 13. Jahrhundert von Roger Bacon verfasst worden, behauptet nun eine Forschergruppe vom *Chicagoer McCrone Institute* und von der *University of Arizona*, ihre Messung des C-14-Gehalts des VM-Pegaments würde den Entstehungszeitraum auf die erste Hälfte des 15. Jahrhunderts einschränken (4.1.3).

4.1.1 Das Voynich-Manuskript im 20. Jahrhundert

In das Zeitalter der Moderne trat das VM Anfang des zwanzigsten Jahrhunderts als Wilfrid Voynich, ein polnischstämmiger amerikanischer Archivar, es nach eigenen Angaben 1912 in der nahe Rom gelegenen Villa Mondragone den dort heimischen Jesuiten abkaufte. Voynich fand es in einer Truhe mit allerlei kostbar kolorierten Handschriften. „It was such an ugly duckling compared with the other manuskripts, with their rich decorations in gold and color, that his interest was aroused at once." (Newbold & Kent 1928:29)

Voynich erstaunen vor allem die bis zum heutigen Tage völlig unentschlüsselten Chiffren sowie die schlichten, aber in ihrer Vielzahl und Unnatürlichkeit bemerkenswerten Zeichnungen seiner Entdeckung. Er beschließt, die gesamte Sammlung zu erwerben, ein Akt, der allerdings nicht exakt belegt werden kann: Als sich ein halbes Jahrhundert später das VM im Besitz des erfolgreichen Buchhändlers H. P. Kraus befindet, fährt dieser zum Zwecke eigener Nachforschungen in den Vatikan. Dabei fällt zur beiderseitigen Verwunderung auf, dass das geheimnisvolle „Cipher-Manuskript" als noch zur Bibliothek gehörig geführt wird, wo es aber aus naheliegenden Gründen – es befand sich ja im Besitz von Kraus – nicht auffindbar war.[1]

Wie das Buch von Voynich zu Kraus kam, ist schnell und ohne Lücken rekonstruierbar: Nach dem Tode Voynichs 1931 vererbt er das VM an seine Frau Ethel und seine langjährige Sekretärin Anne Nill, welche nach dem Tod Ethels schließlich den finanziellen Offerten des Buchhändlers Kraus nachgibt, dessen Erfolg sich hinsichtlich des VM aber eher bescheiden ausmacht: Er erstand es für 24.500 $, wohl auch in der Hoffnung, dass es sich entziffern und mit beträchtlichem Gewinn weiterverkaufen ließe. Stattdessen wurde es zum Ausgestoßenen, zum Paria der Wissenschaftsgeschichte und so überließ es Kraus der Yale-Universität, wo es bis zum heutigen Tage in seiner Vitrine auf denjenigen harrt, der ihm sein Rätsel entringen kann.

4.1.2 Das Voynich-Manuskript im 17. Jahrhundert

Lange Zeit galt die Existenz des VM in der Zeit vor 1912 keinesfalls als bewiesen, da die Yale-Universität die seit 1946 bekannte Radiokarbonmethode – aus welchen Gründen auch immer – bis vor kurzem zur Analyse des Manuskripts nicht zuließ. Voynich selbst

1 Vgl. Kraus 1978. Diese Darstellung widerspricht allerdings einer Passage aus einem Brief von Kraus an William F. Friedman aus dem Jahr 1962 (vgl. http://voynich.nu/sources.html#krau, zuletzt aufgerufen am 11.10.2011). Das Wissen darum, welche Version der Geschichte die richtige ist, hat Kraus wohl mit ins Grab genommen.

wurde verdächtigt, das Manuskript aus finanziellen Interessen gefälscht zu haben. Wie im nächsten Kapitel erläutert, haben sich die Dinge in Hinsicht auf die Anwendung der Radiokarbonmethode offensichtlich geändert, so dass Voynich mit einiger Sicherheit als Fälscher ausscheidet (vgl. 4.1.3). Aber schon vor diesen Entwicklungen gab es Anhaltspunkte, die nahelegten, dass das Manuskript bereits am Prager Hof Rudolfs II (1576 – 1612 Kaiser des Heiligen Römischen Reiches deutscher Nation) zirkulierte: Zum einen findet sich auf der ersten Seite des VM eine Unterschrift eines gewissen Jacobi de Tepenec, seines Zeichens Hofapotheker des erwähnten Kaisers, der seinen adeligen Namen – bürgerlich hieß er Horčický – erst 1608 verliehen bekam und der 1622 starb. Sollte diese Unterschrift authentisch sein[2] ist sie der älteste Hinweis auf die Existenz des VM, es muss dann irgendwann in der Zeit zwischen 1608 und 1622 in Prag bei Tepenec vorgelegen haben.

Die Existenz zweier Briefe, die offenbar beide das VM zum Thema haben,[3] lässt ebenfalls darauf schließen, dass das VM im frühneuzeitlichen Prag vorlag. Einer dieser Briefe wurde von Voynich zusammen mit dem Manuskript gefunden, ein anderer, älterer, erst viel später durch René Zandbergen in einem Prager Archiv entdeckt (vgl. Zandbergen 2011d); gerade dieses zweite Schriftstück liefert starke Hinweise darauf, dass Voynich das VM nicht selbst verfasst hat, insbesondere weil dieser Brief erst nach dem Tod des Buchhändlers gefunden wurde. Beide erwähnten Briefe richten sich aus Prag an den Jesuiten Atanasius Kircher in Rom, eine Art Universalgelehrten zur damaligen Zeit. Dieser hatte sich mit der (in späterer Zeit nicht aufrecht zu erhaltenden) Entschlüsselung von ägyptischen Hieroglyphen einen Namen gemacht und wurde nun von der offenbar an der Decodierung des VM gescheiterten Prager Gelehrtenfraktion um Georg Baresch und Johannes Marcus Marci (die Absender der beiden erwähnten Briefe) dazu auserkoren, ihre Arbeit fortzuführen. Während Baresch sich offenbar noch selbst mit dem VM herumschlug und Kircher in den 1630er Jahren lediglich Kopien des Manuskriptes zukommen ließ, schickte Bareschs Nachlassverwalter Marci nach dessen Tod Kircher das Original zu. Der sonst eher auskunftsfreudige Kircher aber erwähnt den Erhalt des VM geschweige denn eine Beschäftigung damit mit keinem

2 Woran es gewisse Zweifel gibt. Kennedy & Churchill (2004:252) etwa halten die Geschichte, wie Voynich selbst die Unterschrift durch ein zufälliges Mißgeschick entdeckte, für relativ unglaubwürdig (ihm ist angeblich versehentlich Entwicklerlösung über die erste Seite des Manuskripts gelaufen, worauf das Signum erst erkennbar geworden sein soll). Auf der anderen Seite deutet der Vergleich der Unterschrift mit anderen Dokumenten, die de Tepenec unterzeichnete, darauf hin, dass sie von derselben Person stammen, vgl. Hurych (2007).

3 Es ist anfechtbar, dass sich die Briefe tatsächlich auf das VM beziehen, von den meisten Forschern wird es aber nicht ernsthaft bezweifelt. Weiterführende Hinweise zu den beiden Briefen finden sich im Anhang E.

Wort, was Zweifel daran aufkommen lässt, dass er das VM auch tatsächlich erhalten hat. Auf der anderen Seite hat man den Weg des VM als mögliches Exponat des Museum Kircherianum[4] in die Villa Mondragone, wo es dann schließlich Voynich in die Hände fiel, plausibel nachzeichnen können. Für Kirchers als Besitzer des Manuskripts spricht außerdem, dass es in der gleichen Sammlung wie die Kircher'sche Korrespondenz gefunden wurde, und zwar in der Privatbibliothek des ehemaligen Generals der Jesuiten, Pierre Jean Beckx (1795-1887).

4.1.3 Die neueste zeitliche Verortung des Voynich-Manuskripts im frühen 15. Jahrhundert

Der Brief von Marci an Kircher erwähnt, dass (einem gewissen Dr. Rafael[5] zufolge) Rudolf II glaubte, das VM stamme aus der Feder Roger Bacons, eines mittelalterlichen Franziskanermönchs. Diesem Glauben hing auch Voynich an und fand im Mediävisten William Newbold einen Partner, der genau diese These durch seine (wie sich später herausstellte, auf ganzer Linie unhaltbaren) Entschlüsselungen stützte (vgl. 4.3.1). Bis vor kurzem wurde Bacon immer wieder als Verfasser angenommen (so findet sich auch in der Beschreibung der Beinecke-Bibliothek sein Name), der Weg aus dem Nordwesten Englands nach Prag wurde rekonstruiert, bei dem zwei teilweise zwielichtige Gestalten der frühen Neuzeit, John Dee und Edward Kelley, als Überbringer eine wichtige Rolle spielen. Auch Letztere wurden oft als die eigentlichen Verfasser (bzw. Fälscher, vgl. Kapitel 4.3.3) des VM auf den Plan gebracht, letztlich gab es aber keinen handfesten Beweis der Existenz des VM vor 1608 bzw. 1622, der Zeit also, in der de Tepenec sich auf der ersten Seite verewigte.

Dies hat sich jetzt offenbar durch die 2009 erfolgte Datierung mittels materialwissenschaftlicher Prüfung des Manuskripts geändert. Unglücklicherweise gibt es noch immer keine Veröffentlichung zu dieser Prüfung, sie wurde lediglich in einer Dokumentation aus der populärwissenschaftlichen Reihe *Universum* des österreichischen Rundfunks (ORF) erwähnt. Sämtliche zugänglichen Informationen stammten vorher letztlich aus Bezügen zu der ORF-Sendung vom 10.12.2009, so dass die Ergebnisse nicht überprüft werden

4 Der Grundstock dieses Vorläufers eines wissenschaftlichen Museums in Rom wurde aus Kirchers persönlicher Sammlung gebildet, weshalb es auch dessen Namen trug.

5 Dieser wurde inzwischen mit hoher Wahrscheinlichkeit als der böhmische Sprachgelehrte Dr. Rafael Mnishkovski/Minisosky/Missowski identifiziert. Bemerkenswert im Kontext dieser Arbeit ist, dass genau dieser Dr. Rafael die trithemische Polygraphia (vgl. 5.3.2), welche Vorlage für Kirchers *Polygraphia Nova et Universalis* (1663) war, ins Tschechische übertrug (1628), vgl. Embach (2003:254).

können. Wiederholte Versuche, die beteiligten Forscher vom Chicagoer McCrone Institut oder den in der Dokumentation auftretenden VM-Forscher Rene Zandbergen zur schriftlichen Preisgabe der Untersuchungen zu bewegen, blieben bisher ohne Erfolg (vgl. Schmeh 2010). Inzwischen hat die University of Arizona (UoA) zumindest eine Pressemitteilung herausgegeben, welche die gewonnenen Erkenntnisse bestätigt. Dennoch entbehren die nachfolgenden Überlegungen der soliden wissenschaftlichen Grundlage, da auch die in Kapitel 2.3 erwähnte wissenschaftliche Methode über Peer Review hier nicht eingehalten werden kann.

Einem Forenbeitrag von Zandbergen[6] ist zumindest zu entnehmen, dass das Pergament auf sein Alter untersucht wurde, nicht die Schrift. Aus der ORF-Dokumentation lässt sich entnehmen, dass insgesamt vier Streifen des Pergamentes getestet wurden, woraus sich mit 95%iger Sicherheit ableiten ließ, dass die zur Pergamentherstellung verwendete Tierhaut zwischen 1404 und 1438 noch Teil lebender Tiere war.

Klar scheint damit, dass der Verfasser nicht vor 1404 gestorben sein kann, zumindest Roger Bacon kommt damit nicht mehr als Verfasser infrage. Ob allerdings alle späteren potentiellen Autoren (damit auch der potentielle Fälscher Wilfrid Voynich) ausscheiden, ist zumindest zweifelhaft. Zwar erwähnt die Dokumentation weitere materialwissenschaftliche Prüfungen der Tinte, jedoch ohne sie genauer zu belegen. Zusammensetzung und Verwendung deuteten demnach auf eine Beschriftung im 15/16. Jahrhundert hin.[7] Pergament wurde meist umgehend nach seiner Herstellung beschriftet, da das Material – auch durch seine aufwendige Bearbeitung – sehr kostbar war.[8]

Wie erwähnt, ist es relativ schwierig, sich mit Hypothesen auseinanderzusetzen, die einzig in Fernsehdokumentationen aufgestellt und überprüft wurden. So ist nicht klar, auf welcher Basis die 0.95-Evidenz für die Verortung in den ersten vier Jahrzehnten des 14. Jahrhunderts gewonnen wurde. Die Radiokarbonmethode ist bei solch (in Anbetracht der

6 http://voynich.tamagothi.de/2010/01/11/warum-es-hier-momentan-so-still-ist/#comment-2319, zuletzt aufgerufen am 20.05.2011

7 Dies wird so auch von Greg Hodgins von der UoA bestätigt: Eine Altersbestimmung der Tinte sei möglicherweise mit den heute zur Verfügung stehenden Mittel nicht möglich, da die Menge des organischen Materials für eine C14-Analyse zu gering sei. Die verwendeten Farben hätten aber zu Zeiten der Renaissance zur Verfügung gestanden. Quellen dieser Information sind ein ScienceBlog vom 11.02.2011 (`http://scienceblog.com/42652/`) sowie das wissenschaftliche News-Portal *eurekalert* (`http://www.eurekalert.org/pub_releases/2011-02/uoa-uae021011.php`), beide zuletzt aufgerufen am 11.10.2011.

8 Teilweise wurden Pergamente auch mehrfach beschriftet, indem die ursprüngliche Schrift ausgekratzt oder -gewaschen wurde. Solche wiederbeschriebenen Pergamente werden Palimpsest genannt. Es gilt aber als realtiv sicher, dass der VM-Text die erste Beschriftung der Folia ist.

Erdzeitalter) jungen Datierungen ein recht fehleranfälliges Instrument, was am schwankenden C-14-Gehalt der Atmosphäre liegt. Die Schwankung kann auf die unterschiedliche Aktivität der Sonne zurückgeführt werden, die v.a. am Ende des Mittelalters, im frühen 16. Jahrhundert, die Fehlergrenze der Radiokarbonmethode auf +/- 50 Jahre und darüber legen kann (vgl. Fuchs *et al.* 2001). Ab 1650 will sie kein Forscher mehr zur Datierung heranziehen, weil durch das Aufkommen der Industrialisierung der Kohlenstoffgehalt der Luft extrem zunimmt.[9]

Die ORF-Dokumentation folgert darüber hinaus aus der zeitlichen auch eine räumliche Verortung: Da sich auf dem großen, ausklappbaren Folio (vgl. 4.2.1) eine Burg mit sogenannten Schwalbenschwanzzinnen befindet, die zur fraglichen Zeit – im frühen 15. Jahrhundert – lediglich in nördlichen Gegenden Italiens zu finden waren, wird gefolgert, dass das Manuskript genau dort entstanden sein muss. Das widerspricht der Verwendung von beidseitig beschriebenen Pergament (vgl. 4.2), das eher in nördlichen Gegenden (Deutschland, England) verwendet wurde. In Italien war die Verwendung des südlichen, aus Schafshaut bestehenden und nur auf einer Seite geschliffenen und mithin nur auf dieser Seite beschreibbaren Pergaments üblich (vgl. Santifaller 1932).

Durch eine wissenschaftliche Veröffentlichung der Ergebnisse der materialwissenschaftlichen Analyse wären einige offene Fragen zu klären, bis dahin kann wahrscheinlich nur ausgeschlossen werden, dass das VM vor dem 15. Jahrhundert verfasst wurde.

4.2 Beschreibung des Voynich-Manuskripts

Das VM liegt in Form eines Kodex vor, für den mehrere Lagen gefalteten Pergaments zusammengeheftet wurden. In seinem heutigem Zustand weist der Kodex 18 Lagen auf, die vorzugsweise aus vier gefalteten und ineinandergelegten Bögen, also 8 Blättern, bestehen, von denen sowohl Vorder- als auch Rückseiten beschrieben sind. Daneben enthalten einige Lagen eine abweichende Anzahl von Bögen (manche nur einen, andere bis zu sieben) und einige Blätter sind mehrfach gefaltet (sogenannte Foldouts), so dass bspw. die Lage neun zwar nur aus einem einzigen Bogen besteht, der aber ganze sechs Seiten (statt normalerweise zwei) enthält. Völlig aus der Reihe fällt Lage 14, die ausgebreitet eine Fläche von 2*4 Seiten einnimmt (zu sehen auf Abbildung 4.2 auf Seite 89).

Die Seiten haben etwa die Größe eines DIN-A5 Blattes (22,5 * 16 cm) und sind durchgehend auf der Vorderseite paginiert, wobei allgemein vermutet wird, diese Zählung sei

9 Fuchs, pers. Kommunikation.

erst nach Entstehung des Manuskripts hinzugefügt worden (vgl. Zandbergen 2010). Das Fehlen der Seitenzahlen 12, 59-64, 74, 91/92, 97/98 und 109/110 lässt vermuten, dass dem VM im Laufe der Zeit diese Seiten verloren gegangen sind. In der VM-Forschung haben sich diese lückenhaften Folianummern für den Austausch über Positionen im Manuskript durchgesetzt, dabei wird den Zahlen jeweils ein *f* für *folio* vorangestellt. Die Vorderseite wird mit *r* für *recto*, die Rückseite mit *v* für *verso* bezeichnet. Bei Foldouts werden die einzelnen Seiten mit fortlaufenden Nummern versehen, auf der Vorderseite von links nach rechts, auf der Rückseite von rechts nach links. Die Bezeichnung *f1r* etwa verweist auf die Vorderseite des ersten Folio, *f67v2* auf die Rückseite des Folio 67, zweite Seite von rechts.[10]

Der unbekannte Autor (oder auch die unbekannten Autoren) des VM füllten die Seiten sowohl mit Zeichnungen als auch mit einer sehr speziellen Art von Text. Die Grafiken sind sehr unterschiedlichen Typs – es finden sich u.a Pflanzen, Tierkreiszeichen und in seltsamen Gefäßen badende Frauen. Weil noch kein Zugang zum Text gefunden wurde, wird das VM traditionell (u.a. im Katalog der Beinecke Library) anhand dieser Abbildungen in verschiedene Sektionen eingeteilt, was in 4.2.1 näher erläutert wird. Kapitel 4.2.2 geht auf die Besonderheiten der Schrift und den einzigartigen Zeichensatz des VM ein, bedeutsame statistische Eigenheiten des Textes werden in 4.2.3 thematisiert.

4.2.1 Zeichnungen und Gliederung

Von einigen wenigen Ausnahmen abgesehen sind die Seiten des VM voll von großformatigen Zeichnungen, die mit Tinte ausgeführt und nachträglich koloriert wurden. Es wird im Allgemeinen davon ausgegangen, dass die Zeichnungen vor dem Text zu Papier gebracht wurden,[11] da sie des öfteren in den Text hineinragen oder gleichsam von ihm umflossen werden. Bisweilen findet sich der Text auch innerhalb der Zeichnungen als eine Art Beschriftung.

Im Gegensatz zum Text sind die Zeichnungen interpretierbar, zumindest lassen sich verschiedene Typen von Zeichnungen unterscheiden. Die Gliederung des VM in fünf bis sechs Teile resultiert aus dieser Typisierung der Zeichnungen. Sie ist bereits früh vorgenommen worden, in die Manuskriptbeschreibung der Beinecke Library eingegangen und – bis auf

10 Eine sehr übersichtliche grafische Darstellung der Anordung einzelner Folia im Kodex findet sich auf Zandbergens Webseite (`http://voynich.nu/folios.html`), zuletzt aufgerufen am 11.10.2011.

11 Wie so vieles im Zusammenhang mit dem VM ist auch diese Behauptung nicht endgültig zu klären. Schwerdtfeger (2006) etwa weist darauf hin, dass auf Folio 81r der Text am rechten Rand auf seltsame Art formatiert ist, als wenn dort Platz für eine weitere Zeichnung gelassen worden wäre.

Abbildung 4.1: *Folia der unterschiedlichen VM-Sektionen. Oben links f66v aus der botanischen Sektion; oben rechts f71r mit Tierkreiszeichen aus der astrologischen Sektion; unten links f101v aus der pharmazeutischen Sektion; unten rechts f78r aus der balneologischen Sektion.* Quelle: Yale University, Beinecke Rare Book and Manuscript Library.

die Neuzuordnung einzelner Seiten, s.u. – auch heute noch allgemein als Struktur des VM anerkannt. Die äußeren Seiten des Kodex, also f1r und f116v, werden hier aus der Gliederung ausgenommen, weil sie in mehrfacher Hinsicht auffällig sind: Auf der ersten Seite findet sich ein in vier Paragraphen[12] aufgeteilter, inzwischen stark verblasster Text und keine Zeichnung; die letzte Seite ist – im Gegensatz zum restlichen VM, wo die Seiten fast vollständig gefüllt sind – zu fast 4/5 leer, nur ganz am oberen Rand finden sich vergleichsweise winzige Zeichnungen und ein kurzer Textabsatz, der oft auch als Schlüssel für die Dechiffrierung des Gesamttextes angenommen wurde. Zwischen diesen Außenseiten des VM erstreckt sich die folgende, an der Art der Zeichnungen orientierte Gliederung:

1. **Pflanzenkundliche bzw. botanische Sektion**: Dieser Teil des Manuskripts ist mit Abstand der umfangreichste und stellt über die Hälfte des gesamten Kodexumfangs. Obwohl es bisher nicht gelungen ist, die Abbildungen dieses Teils tatsächlich existierenden Gewächsen zuzuordnen,[13] sind sie doch durch ihren Aufbau Wurzel - Stängel - Blätter - Blüten eindeutig als Pflanzen zu klassifizieren. D'Imperio (1978b:15) vermutet, dass es dem Zeichner weniger um möglichst naturnahe Abbildung von Objekten ging, als um die Anfertigung mechanisch-struktureller Skizzen. Eine typische Seite der botanischen Sektion enthält eine abgebildete Pflanze und bis zu drei Paragraphen Text, entweder über der Pflanze oder um diese herumlaufend (vgl. Abbildung 4.1 auf Seite 85, links oben). Die Sektion erstreckt sich über die gesamte erste Hälfte des Kodex, f1v bis f66v (wobei f12 und f59-f64 fehlen und f66r nur Text, aber keine Pflanze enthält), nach neuerer Zählung kommen noch Seiten aus den ursprünglich dem pharmazeutischen Abschnitt zugeordneten Seiten hinzu (s.u.).
2. **Astronomische bzw. astrologische bzw. kosmologische Sektion**: Diese Sektion – die oftmals auch in die drei hier zusammengefassten Abschnitte unterteilt wird – besteht im Gegensatz zum botanischen Teil teilweise aus Foldouts. Die Abbildungen setzen sich aus konzentrischen Kreisen zusammen, die teils mit Text, teils mit Darstellungen von Sternen oder auch solchen von mitunter nackten, bisweilen aber auch vollständig bekleideten, in gefüllten Zubern sitzenden oder auch stehenden

12 Zur Nomenklatur der textuellen Einheiten vgl. 4.2.2.

13 Es gab immer wieder Versuche, die Pflanzen zuzuordnen, einer der ausführlichsten lässt sich bei Scott (2008) verfolgen. Kontrovers diskutiert wurden v.a. die Identifizierung von Sonnenblume (f93r) und Spanischem Pfeffer (f100r), beides Neuweltgewächse. Sollten diese beiden Zuordnungen wirklich zutreffen, so wäre dies ein starker Hinweis darauf, dass das VM frühestens entstanden sein kann, *nachdem* Kolumbus von seiner ersten Amerika-Reise wiederkehrte.

Frauen versehen sind. In der Mitte der Abbildungen finden sich stilisierte Himmelskörper (mitunter tragen sie Gesichter, der Abschnitt wird oft als astronomischer bzw. kosmologischer bezeichnet) und Tierkreiszeichen (der astrologische Teil, vgl. Abbildung 4.1 auf Seite 85, rechts oben). Die Sektion erstreckt sich durchgehend von f67r bis f73v. Das Fehlen der Tierkreiszeichen Wassermann und Steinbock lässt vermuten, dass das fehlende Folio 74 diese abbildete und damit ebenso zur astrologischen Sektion gehörte. Zusammenhängende Textblöcke finden sich nur vereinzelt, der Text innerhalb der Kreise ist entweder konzentrisch oder – seltener – radial angelegt und passt sich längenmäßig nahezu perfekt in die Zeichnungen ein. Erwähnenswert sind noch die in (lateinischem) Klartext an die Tierkreiszeichnungen geschriebenen Monatsnamen, von denen allerdings angenommen wird, dass sie – wie die Foliozählung – nicht vom originalen Autor, sondern von einem späteren Besitzer hinzugefügt wurden. Zur kosmologischen Sektion wird außerdem noch das oben schon erwähnte, 2*4 Seiten große Foldout f85/86 gezählt, das auf der Vorderseite neun auf einer quadratischen Grundstruktur angelegte, miteinander verbundene Rosetten abbildet.[14]

3. **Balneologische bzw. biologische bzw. anatomische Sektion**: Dieser Teil wird meist als der bemerkenswerteste und rätselhafteste bezeichnet. Eine große Zahl nackter, ausschließlich weiblicher Figuren mit gewölbten Bäuchen sitzen oder stehen meist in Wannen, Becken und ähnlichen Gefäßen, die über komplexe Rohrsysteme miteinander verbunden sind. Die Rohrsysteme sind mit diversen End- und Verbindungsstücken versehen, die teils organisch anmuten, teils an mechanische Gegenstände erinnern. Die Ungewöhnlichkeit dieser Darstellungen lässt immer wieder Spekulationen darüber aufkommen, die Zeichnungen stellten nicht allein badende Frauen dar, sondern müssten im übertragenen Sinn interpretiert werden, vgl. etwa Kennedy & Churchill (2004:19f) “Die Illustrationen haben eindeutig sexuellen Charakter (...) weil sie an Fruchtbarkeit, Geburt und Tod denken lassen. Samen oder Pollen quellen aus Röhren, traubenartige Eibündel erzeugen Flüsse, und die Flüsse ergießen sich in Teiche, in denen Nymphen baden und Tiere ihren Durst stillen Einige Bilder lassen sogar an moderne Abbildungen von menschlichen Eierstöcken und Eileitern denken.” Im Vergleich zu den oben beschriebenen Sektionen findet sich im balneologischen Abschnitt (der sich von f75r bis f84v erstreckt) sehr wenig

14 Eine sehenswerte dreidimensionale Animation der Rosettenseite wurde von H. Richard SantaColoma unter `http://www.youtube.com/watch?v=NGrNIBOsHYk` (zuletzt aufgerufen am 11.10.2011) veröffentlicht.

Leerraum auf den Seiten, es gibt durchgehend mehr Text als im pflanzlichen Teil, dieser schmiegt sich außerdem immer eng an die Zeichnungen an (vgl. Abbildung 4.1 auf Seite 85, rechts unten).

4. **Pharmazeutische Sektion**: Vom Füllgrad der Seiten her erinnert die pharmazeutische Sektion an die balneologische, im Unterschied zu dieser findet sich hier allerdings weniger Text, der zudem meist in rechteckige Äbsätze geliedert ist (sich also nicht an die Zeichnungen anschmiegt). Der Hauptunterschied aber besteht darin, dass sich auf den Seiten keine badenden Frauen, sondern mehrere kleinere Abbildungen verschiedener Pflanzen neben bunten, offenbar zur Aufbewahrung dienenden Gefäßen befinden (vgl. Abbildung 4.1 auf Seite 85, links unten). Pflanzen und Gefäße sind darüber hinaus beschriftet, so dass dieser Teil stark an ein Rezeptbuch erinnert. In der ursprünglichen Gliederung erstreckte sich die Sektion von 87r bis 102v, (f97/f98 fehlen) inzwischen werden aber die Seiten f87, f90 und f93-96, die jeweils nur eine einzelne, großformatige Pflanzenzeichnung tragen, zum pflanzenkundlichen Abschnitt gezählt.
5. **Sektion mit kontinuierlichem Text**: Wird auch Sterne- oder Rezept-Sektion genannt, da am linken Rand des ansonsten die ganzen Seiten füllenden Fließtextes Sterne unterschiedlichen Stils gezeichnet sind. Auch in dieser Sektion fehlen zwei Folia (f109/f110), die sich wahrscheinlich auf einem Bogen befunden haben. Die Sektion erstreckt sich von f103r bis f116r, bestand also ursprünglich aus 27 Seiten Fließtext, von denen 23 verblieben sind. Die Sterne erinnern an Aufzählungszeichen und sind – soweit man das erkennen kann – einzelnen Paragraphen zugeordnet. Eine Ausnahme dabei bildet die letzte halbe Seite der Sektion (f116r), wo mehrere Paragraphen nicht mit einem Stern versehen sind. Die meisten kryptoanalytischen Angriffe wurden auf Basis dieser textuell sehr reichen Sektion gemacht, gemäß dem Leitgedanken, Chiffren umso besser angreifen zu können, je mehr verschlüsselter Text vorliegt (vgl. Kapitel 5.2). Wie eingangs erwähnt, war keiner der Versuche bislang erfolgreich.

Diese Arbeit hat zum Ziel, das VM als Anwendung eines Labors für die Textprozessierung zu betrachten, weshalb die Beschäftigung mit den Zeichnungen lediglich der Übersicht über den Aufbau des Manuskripts galt. So wird die obige knappe Darstellung der Abbildungen des Manuskripts der Mannigfaltigkeit der Zeichnungen, die im VM auftreten, nicht im Mindesten gerecht. Jede der abgebildeten Pflanzen hat ihre Besonderheit, jedes Sternbild ist von ganz unterschiedlichen Satelliten auf den umliegenden konzentrischen

Abbildung 4.2: *Weitere Folia; links eine Seite aus der Sektion mit kontinuierlichem Text; rechts das 2*4 Seiten große Foldout f85/86 (relativ zu den anderen Seiten verkleinert).* Quelle: Yale University, Beinecke Rare Book and Manuscript Library.

Kreisen versehen, jede Seite der balneologischen Sektion wartet mit völlig neuartigen Badeapparaturen auf. Bisweilen wirkt es, als hätte der Autor/Zeichner eigens kleine Eigentümlichkeiten eingeflochten, die den Betrachter unterhalten oder nur weiter in die Irre führen sollen. Als Beispiele seien hier der kleine Drache genannt, der an der Pflanze von f25v knabbert oder die Burg mit den Schwalbenschwanzzinnen auf einer der Rosetten der großen, ausklappbaren Lage f85/86.[15]

4.2.2 Schrift und Zeichensatz

Im Kontext dieser Arbeit ist der Text der essentielle Teil des VM. Auf den ersten Blick wirkt er für das in Betrachtung mittelalterlicher Handschriften geübte Auge nicht einmal sonderlich fremdartig:

> "At first glance, the text that is the heart of the mystery appears to be no problem at all. It does not look cryptic. It looks like ordinary late-medieval

15 Für detailliertere Betrachtungen einzelner Zeichnungen sei auf den Blog von Elias Schwerdtfeger (`http://voynich.tamagothi.de/`, zuletzt aufgefufen am 11.10.2011) verwiesen.

> handwriting. The symbols preserve the general form of letters of that time, which they are not; they are like old friends whose names are on the tip of one's tongue." (Kahn 1996:863f)

Die Schrift wird allgemein als schwungvoll und fließend bezeichnet, als wäre der Verfasser sehr vertraut mit dem Zeichensatz gewesen und hätte bereits Erfahrung im Abfassen vergleichbarer Dokumente gehabt (D'Imperio 1978b:23), bzw. es scheint, als wäre der Text von einem geübten Schreiber kopiert worden (Kahn 1996:864), da offenbar keine Fehler gemacht oder zumindest keine korrigiert wurden. In einem breit rezipierten, aber unveröffentlichten Aufsatz (Currier 1976) behauptet der Autor, mindestens zwei verschiedene Handschriften im Text des VM erkannt zu haben. Durch statistische Analysen weist er nach, dass diese zwei Handschriften auch verschiedenen „Dialekten" des Voynich'schen entsprechen, da sie sehr unterschiedliche Distributionen ihrer Einheiten aufweisen (weiteres vgl. 4.2.3). Currier folgert daraus, dass das VM von zumindest zwei Autoren geschrieben worden sein muss. Später wird ein noch sehr viel größerer Kreis von Autoren angenommen, allerdings sind sowohl Curriers als auch die auf seiner Arbeit basierenden Analysen unter Handschriftenexperten und damit unter VM-Forschern allgemein umstritten (vgl. Hurych 2008a).

Wie bereits oben erwähnt, erinnern die Zeichen des VM an spätmittelalterliche Handschriften, ohne jedoch wirklich auf diese abgebildet werden zu können. Die größten Analogien bestehen zwischen dem VM und im Mittelalter verbreitet eingesetzter lateinischer Kurzschrift, sowie frühen Adaptionen arabischer Ziffern (vgl. D'Imperio 1978b:23 und v.a. Abbildungen 16/17 auf 94f).

Um den Text des VM maschinell analysieren zu können, ist es notwendig, ihn in diskrete Einheiten eines endlichen Alphabets zu zerlegen. Diese Aufgabe ist im Falle des VM nicht unproblematisch, u.a. weil schon hinsichtlich der Größe des Alphabets Unklarheit besteht: Die verbreitetsten bisher entworfenen Transkriptionsalphabete bestehen aus 23 bis 40 unterschiedlichen Einheiten. Die beiden Hauptprobleme bestehen darin, dass (A) in der Handschrift bisweilen keine klaren Abgrenzungen zwischen einzelnen Zeichen gemacht werden können und (B) sich im Manuskript eine ganze Reihe Zeichen finden, die extrem selten auftreten, manche von ihnen finden sich gar nur einmal. Bei diesen seltenen – auch *Weirdos* genannten – Zeichen ist umstritten, ob es sich lediglich um künstlerische Abwandlungen häufig vorkommender Zeichen oder um Zeichen mit einer eigenen Bedeutung handelt. Behandelt man sie als eigenständige Zeichen, so wird das Transkriptionsalphabet auf den vierfachen Umfang aufgebläht, ordnet man die Weirdos ähnlichen,

aber frequenteren Zeichen zu, so wird eventuell unzutreffend generalisiert.

Auch Problem (A) betrifft die Zuordnung von Glyphen[16] zu abstrakten Zeichen im Sinne von Graphemen[17]. Eine simple 1:1 Zuordnung ist mit Kenntnis lediglich eines Dokuments der Sprache schwerlich möglich: Ist etwa das Zeichen ı als einzelnes Zeichen zu betrachten oder nur als Bestandteil von Zeichenkombinationen? Die First Study Group unter Friedman (kurz FSG) etwa transkribiert ﬓ als *L*, ıﬓ als *N*, ııﬓ als *M*, Currier verfährt genauso. Die aktuellste Transkriptionstabelle des European Voynich Alphabets (kurz EVA) von Zandbergen & Landini (2000) dagegen transkribiert die gleichen Zeichen(kombinationen) zu *n*, *in* und *iin*, auch weil ı in den gleichen Kombinationen mit ﬓ und ﬔ vorkommt. Unklar ist auch der Status der sogenannten *Gallows* (von engl. Gallow – der Galgen; die so bezeichneten Glyphen sind ausgeprägt vertikal ausgerichtet) ﬗ﬘﬙﬚, die sowohl für sich allein stehend, wie auch als Ligatur mit der – daneben auch einzeln auftauchenden Glyphe – ﬕ vorkommen: ﬗﬕ﬘ﬕ﬙ﬕ﬚ﬕ. Diese Ligaturen werden von Currier wieder als einzelne Zeichen transkribiert (*Q, W, X, Y*), während sie bei der FSG in Kombinationen zweier (*PZ, FZ, HZ, DZ*), in EVA in solche dreier Zeichen (*cth, cph, cth, cfh*) überführt werden. Durch die Anlage der Transkriptionsalphabete werden verständlicherweise Statistiken über den Text (bspw. die Wortlängenverteilung) und damit auch seine Analyse beeinflusst.[18]

Zur Verdeutlichung der obigen Ausführungen wurden in der Tabelle 4.1 auf Seite 92 die gängigsten Transkriptionsalphabete zum Voynich-Manuskript zusammengeführt.[19]

Die größten Unterschiede in den Transkriptionalphabeten finden sich

1. bei den stark vertikal ausgeprägten Zeichen, den Gallows (Zeilen 8-11).
2. bei den Kombinationen der Gallows mit dem stark horizontal ausgeprägten Zeichen (Zeile 6) zu den komplexen Ligaturen (Zeilen 12-15).
3. bei den Abfolgen von einem oder mehreren Vorkommen des kurzen schrägen Strichs (durchgehend mit *i* bezeichnet, Zeile 18) mit unterschiedlichen Zeichen am Ende (Zeilen 19-34).

16 Als Glyphen bezeichnet man zusammenhängende graphische Einheiten einer Handschrift.

17 Gemäß der Distributionshypothese sind Grapheme die kleinsten bedeutungsunterscheidenden Einheiten einer Schriftsprache, vgl. Kap. 6.2.1, auch zur konkurrierenden Repräsentationshypothese.

18 Auch in dieser Hinsicht ist der Einsatz eines Systems wie Tesla, in dem unterschiedliche Versuchsanordnungen einander gegenüber gestellt werden können, für einen Anwendungsbereich wie die Analyse des VM von beträchtlichem Vorteil.

19 Die Zusammenstellung wurde zu großen Teilen aus Zandbergen (2011b) übernommen.

	Glyphe	Currier	FSG	Tiltman	EVA	Frogguy
1		4	4	4	q	4
2		O	O	O	o	o
3		8	8	8	d	8
4		9	G	G	y	9
5		2	2	2	s	s
6		S	T	T	ch	ct
7		Z	S	S	sh	c't
8		P	H	H	t	qp
9		B	P	P	p	qi
10		F	D	D	k	lp
11		V	F	F	f	lj
12		Q	HZ	HZ	cth	cqpt
13		W	PZ	PZ	cph	cqit
14		X	DZ	DZ	ckh	clpt
15		Y	FZ	FZ	cfh	cljt
16		A	A	A	a	a
17		C	C	C	e	c
18		I	I	I	i	i
19		E	E	E	l	x
20		G	IE	IE	il	ix
21		H	IIE	IIE	iil	iix
22		1	IIIE	IIIE	iiil	iiix
23		R	R	R	r	2
24		T	IR	IR	ir	i2
25		U	IIR	IIR	iir	ii2
26		0	IIIR	IIIR	iiir	iii2
27		D	L	L	n	v
28		N	N	IL	in	iv
29		M	M	IIL	iin	iiv
30		3	IIIL	IIIL	iiin	iiiv
31		J	K	K	m	g
32		K	IK	IK	im	ig
33		L	IIK	IIK	iim	iig
34		5	IIIK	IIIK	iiim	iiig
35		6	6	6	g	cg
36		7	7	7	j	&
37		(n)	Y	Y	x	n
38		(v)	V	V	v	^

Tabelle 4.1: *Unterschiedliche Transkriptionsalphabete für die VM-Glyphen. Diese Aufstellung wurde teilweise von Zandbergen (2011b) übernommen und erweitert.*

Das Alphabet von Currier hält sich konsequent an die Vorgabe, dass eine Glyphe genau durch ein Zeichen repräsentiert wird, d.h. Ligaturen werden nicht aufgelöst, jede graphische Einheit bekommt genau ein Symbol des Transkriptionsalphabets zugewiesen. Das andere Extrem ist das Frogguy-Alphabet (Guy 1996), das möglichst konsequent versucht, Ligaturen in einzelne Bestandteile zu zerlegen und sie damit in Kombinationen von Symbolen überführt. Die anderen Ansätze befinden sich irgendwo dazwischen: Die von Friedman geleitete FSG etwa löst die komplexen horizontal/vertikal-Ligaturen in zwei Buchstaben auf – allerdings wird dabei zwar der Code des vertikalen, nicht aber der des horizontalen Zeichenteils beibehalten (T wird zu Z, vgl. Zeilen 6 und 12-15). Die ɪ-Kombinationen werden zum großen Teil aufgelöst (Zeilen 20-22, 24-26, 30, 32-34), in zwei Fällen aber nicht (Zeilen 30/31). Die konsequente Behandlung auch letzterer ist die einzige Änderung, die Tiltman (1951) in seinem Alphabet im Vergleich zu dem der FSG einführt.

Das EVA-Alphabet ist der jüngste der vorgestellten Transkriptionsansätze. Es geht ähnlich analytisch an die Auflösung von Ligaturen heran wie das Frogguy-Alphabet, behandelt im Gegensatz zu diesem aber bspw. die Gallows als einzelne Zeichen, nicht als Kombinationen (wobei man hier nicht unbedingt von fehlender Konsequenz ausgehen muss – die Zeichen dieser Frogguy-Gallows treten nämlich ausschließlich in den Gallow-Kombinationen, also nicht als Einzelzeichen auf). Das EVA-Alphabet bietet darüber hinaus eine ganze Reihe von Vorteilen:

1. Es besteht lediglich aus Buchstaben; Aufgrund der geschickten Zuordnung zu Vokalen und Konsonanten sind die EVA-Transkriptionen der VM-Wörter aussprechbar. Damit ist es möglich geworden, sich über den Text des VM auch mündlich auszutauschen.
2. Zur EVA-Transkription liegt sowohl ein *TrueType*-, wie auch ein LaTeX-Font vor, der die Zeichen der EVA-Transkriptionen in VM-Zeichen darstellen kann (letzterer wird für diese Arbeit genutzt).
3. Eine der wichtigsten Ressourcen für die textuelle Analyse des VM, das im Kapitel 6 verwendete Interlinear Archive File (IAF), ist in EVA codiert.
4. Neben dem in der Tabelle 4.1 dargestellten Basic EVA mit 27 Zeichen (Kleinbuchstaben zzgl. Hochkomma) können in EVA auch Satzzeichen und Metainformationen codiert werden. In einer erweiterten Fassung (Extended EVA Characters) sind alle Zeichen des VM, auch lediglich einmalig vorkommende, sowie die Paginierungsziffern aufgenommen worden. Im Gegensatz zu allen anderen Ansätzen lässt sich das VM damit vollständig in EVA abbilden.

5. Nicht zuletzt ist EVA knapp, aber ausreichend und öffentlich zugänglich dokumentiert (Zandbergen & Landini 2000).

4.2.3 Statistische Besonderheiten des Textes

Nicht allein die Form der Zeichen aus dem VM-Text ist ungewöhnlich, auch die Art ihrer Verwendung gibt den Wissenschaftlern Rätsel auf, da diese sich durch gleich mehrere Eigentümlichkeiten signifikant von allen bekannten Schriftsystemen unterscheidet. Es sind diese Eigenheiten, die die Basis für die in 4.3 erörterten Interpretationen bilden. Dieses Kapitel versucht, die – von der Fremdartigkeit der verwendeten Zeichen im letzten Kapitel unabhängigen – Eigenheiten des VM-Textes möglichst vollständig darzustellen. Zunächst werden die Auffälligkeiten hinsichlich Häufigkeit und Verteilung der Zeichen betrachtet, im Anschluss daran die Besonderheiten von Zeichenkombinationen (Wörtern), wobei auch auf die eigentümliche Wortlängenverteilung des VM-Textes eingegangen wird. Als nächstes werden Studien zum inneren Aufbau der VM-Wörter vorgestellt, abschließend werden verschiedene weitere Besonderheiten des VM-Textes aufgeführt.

Zu allererst aber bedarf es der Begriffsklärung, mit welchen (offensichtlich) linguistischen Einheiten wir es im VM zu tun haben. Wie schon in 4.2.2 gezeigt wurde, ist bereits die Definition von Schriftspracheneinheiten oder **Graphemen** nicht ohne weiteres möglich, was zu einer Reihe unterschiedlicher Transkriptionsalphabete führt. Wesentlich einfacher ist die Definition von **Wörtern**: Wahrscheinlich gibt es im VM keine Interpunktion,[20] so dass Wörter immer durch Spatien getrennt sind. Zwar sind einige der worttrennenden Zwischenräume umstritten, dem wird im IAF aber damit begegnet, dass sichere Wortzwischenräume mit einem Punkt, unsichere mit einem Komma ausgedrückt werden. In Ermangelung von Interpunktionszeichen lässt sich nur noch eine weitere linguistische Einheit ausdrücken, indem nämlich Textblöcke voneinander abgesetzt werden. Diese Einheiten werden als **Paragraphen** bezeichnet. Paragraphgrenzen sind im VM durch einen untypisch großen Zwischenraum gekennzeichnet, der auch mit Verkürzung der letzten Zeile korrespondiert;[21] in der Sterne-Sektion werden Paragraphen zudem durch Sterne eingeleitet. Vereinzelte Wörter, die sich einzeln außerhalb von Paragraphen und in der Nähe von Zeichnungen finden, werden als **Labels** bezeichnet.

20 Es ist natürlich möglich, dass Interpunktionszeichen irrtümlich zu den Graphemen gezählt werden. Offenbar existierte einmal ein Aufsatz von Landini zum Thema ("Counting commas in the Voynich Manuskript"), der aber inzwischen nicht mehr auffindbar ist und offensichtlich auch bei gängigen Internet-Archiven (wie archive.org) gelöscht wurde.

21 Die vorliegende Arbeit bedient sich im Übrigen genau der gleichen Strategie.

Zeichenhäufigkeit, Zeichenentropie

Auf die Probleme der Zuordnung von Glyphen zu Graphemen wurde schon in 4.2.1 eingegangen, sämtliche Statistiken über die Zeichen des VM sind naturgemäß abhängig vom gewählten Transkriptionsalphabet: Legt man Transkriptionen von Currier oder Friedman zugrunde, hat man es mit sehr viel mehr Graphemen, aber sehr viel kürzeren Wörtern zu tun, als wenn man eine Transkription nach EVA oder Frogguy für seine Analyse wählt.

D'Imperio (1978b:28) beschränkt sich daher eher auf eine Beschreibung der Glyphen und führt eine ganze Reihe ihrer Eigenschaften auf: Lediglich ı und c kommen in signifikanter Form als Verdoppelungen vor, dafür wird dieses Mittel bei den beiden Zeichen dann auch exzessiv genutzt; bisweilen können sie gar dreifach hintereinander stehen. Bestimmte Grapheme zeigen ein auffälliges Verhalten hinsichtlich ihrer Vorkommen innerhalb von Wörtern, sie stehen etwa vorzugsweise am Wortanfang, in der Wortmitte oder am Wortende, z.B. steht 4 immer vor o und so gut wie immer am Wortanfang. Erste Buchstaben von Paragraphen sind – zumindest in der pflanzenkundlichen Sektion – in fast allen Fällen die Gallows f, p, t, k, die dafür so gut wie nie die Anfangsbuchstaben der Labels bilden. Letztere tendieren dazu, mit o, d, y oder – seltener – mit s oder r zu beginnen.

Die relative Häufigkeit der Grapheme wurde – möglicherweise auch aufgrund der Zuordnungsproblematik – erstaunlich selten untersucht. Eine der wenigen Arbeiten stammt von Hurych (2008b), der eine bemerkenswerte Übereinstimmung der Häufigkeitsverteilung von Graphemen im VM und in lateinischen Texte feststellt. Williams (1999) untersucht lediglich die relative Häufigkeit der Anfangsbuchstaben von fünf Seiten des VM und stellt fest, dass sie mit der relativen Häufigkeit von Anfangsbuchstaben eines alten griechischen Lexikons korrelieren. Er schließt daraus, dass es sich beim VM-Text um einen Code (vgl. 5.1) handelt, der sich an griechischen Wörtern orientiert. Guy (1991a) wendet den von ihm selbst ins Englische übertragenen Sukhotin-Algorithmus[22] auf den VM-Text an und ermittelt insgesamt sechs Vokale (c, o, g, a und ı sowie die Glyphen-Kombination cc). Dazu versucht er, basierend auf der Distribution von Zeichen, Allographen[23] zu detektieren. So weisen cc und a etwa eine nahezu komplementäre Distribution ihrer Nachfolger

22 Zuerst wurde der Algorithmus von Sukhotin 1962 vogestellt, später von Guy (1991b) in der Zeitschrift *Cryptologia* aufgegriffen. Der Algorithmus basiert auf der Annahme, dass Vokale dazu tendieren, eher neben Konsonanten, als neben anderen Vokalen zu stehen. Inzwischen wurde dieser sehr einfache, fünf-schrittige, iterative Algorithmus an mehreren Sprachen getestet, wobei festgestellt wurde, dass er umso genauer arbeitet, je exakter die Phoneme durch Grapheme abgebildet sind und je weniger diskrete Vokal-Grapheme existieren (Sassoon 1992).

23 Genauso wie *Allophone* unterschiedliche Varianten (Phone) des gleichen Phonems sind, sind *Allographe* unterschiedliche Varianten (Glyphen) des gleichen Graphems.

auf, was darauf hindeutet, dass sie für dasselbe Graphem in unterschiedlichen Kontexten stehen könnten.

Eine ganze Reihe von Untersuchungen wurden zur Entropie[24] des VM-Textes durchgeführt, die erste von Bennett (1976) im Rahmen einer allgemeinen Einführung in die Verwendung des Computers für die Berechnung komplexer Werte. Bennett stellte fest, dass im VM-Text vor allem die Entropie höherer Ordnung, auch Verbundentropie genannt, signifikant niedriger war als die Verbundentropie aller bekannten natürlichen Sprachen.[25] Konkret berechnete er für die ersten 10 Seiten des VM h_1 mit 3,66, h_2 mit 2,22 und h_3 mit 1,86. Verglichen mit den Werten für europäische Sprachen (bspw. Lateinisch aus einer nicht näher bezeichneten Schrift von Juluis Cäsar $h_1 = 4,05$, $h_2 = 3,05$, $h_3 = 2,38$) sah Bennett eine Näherung von $(h_n)_{Voynich} \approx (h_{n-1})_{europaeischeSprachen}$. Die Entropie eines VM-Zeichens ist also signifikant niedriger als die natürlicher Sprachen und entspricht im Ungefähren der Verbundentropie zweier Zeichen in natürlichen Sprachen. Niedrige Entropiewerte sind ein Indikator dafür, dass der Text sehr redundant und informationsarm ist. Dies muss von einer Theorie, auf deren Grundlage der Text analysiert werden soll, erklärt werden, denn gängige Verschlüsselungsverfahren weisen keine derartigen Abweichungen des Informationsgehalts auf.

Bennetts Vergleich von VM-Text und natürlichsprachlichen Texten ist insofern problematisch, als dass auch für die Berechnung der Verbundentropie gilt, dass der maximale Entropiewert stark vom Umfang des Zeichenalphabetes abhängt. So bestand Bennetts Grapheminventar bei der Errechnung der Verbundentropie des Hawaiianischen lediglich aus 13, sein – dem EVA-Schema ähnelndes – verwendetes Transkriptionsalphabet aber aus 20 Elementen. Stallings (1998) hält in einem solchen Fall den Vergleich von h_x verschiedener Sprachen für nicht aussagekräftig. Stattdessen könne man die Werte von Sprachen mit unterschiedlich vielen Graphemen durch die Differenz $h_1 - h_2$ viel besser vergleichen. Diese Differenz ist bei dem von Stallings analysierten pflanzenkundlichen Abschnitt des VM mit 1,53 (bei zugrundeliegender Currier-Transkription mit 34 Graphemen) bzw. 1,78 (bei zugrundeliegender EVA-Tanskription mit 21 Graphemen) mehr als doppelt so hoch, wie für Textproben anderer Sprachen (Latein: 24 Grapheme, 0,70; Hawaiianisch: 13, 0,75; Englisch 27, 0,95). Einigermaßen nahe kommt dem Wert nur ein Text in japanischer Silbenschrift (Kana; 71 Grapheme, 1,37). Auch der Versuch, bei der Analyse

24 Für eine Einführung in die Berechnung verschiedener Entropie-Werte vgl. Anhang B.

25 Die einzige von Bennett untersuchte Sprache, die den niedrigen h_2 und h_3-Werten des VM nahekam, war das Hawaiianische aus der polynesischen Sprachfamilie, die allerdings mit an Sicherheit grenzender Wahrscheinlichkeit dem Verfasser des VM nicht bekannt war.

extrem repetitiven Texts ähnlich hohe Werte zu erreichen, scheitert, da alle von Stallings untersuchten Texte Werte von unter 1 aufweisen. Stallings sieht durch die Abweichung der Werte als erwiesen an, dass der Text des VM keine Verschlüsselung irgendeiner natürlichen Sprache sein kann, die dem Muster entspricht, dass eine Klartexteinheit immer durch genau eine Geheimtexteinheit ersetzt wird (das heißt, dass Klar- und Geheimtext ungefähr die gleiche Länge haben). Stattdessen deutet die hohe Repetitivität darauf hin, dass der zugrundeliegende Klartext – so es ihn geben sollte – sehr viel kürzer als der Text des VM sein müsste. Zur Untermauerung dieser These verschlüsselt Stallings ein lateinisches Textstück mit einer Chiffre, welche die Hälfte der Klartextalphabets durch ein einzelnes Zeichen ersetzt, die andere Hälfte durch eine Reihe von sechs bis acht Zeichen.[26] Tatsächlich beträgt bei einem so verschlüsselten Text $h_1 - h_2$ 1,60, der durchaus mit dem VM-Werten vergleichbar ist. Abgesehen vom gleichen Entropiewert hat die Chiffre aber nicht viel Ähnlichkeit mit dem VM-Text, wie Stallings (1998: Tabelle 5) selbst bemerkt: "However, it's clear that this is not the same pattern as Voynich text. It might be best to look for patterns subjectively." Das liegt daran, dass die Wörter des VM eine interne Struktur haben, die im übernächsten Abschnitt thematisiert wird. Ein Verschlüsselungsverfahren, das als dem VM zugrundeliegend angenommen werden soll, muss also sowohl den ermittelten Entropiewerten, als auch den Wortbildungsmustern gerecht werden.

Wortlänge, Worthäufigkeit, Wortentropie

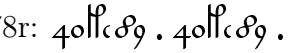
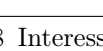
Bei der quantitativen Analyse von Texten wird immer gerne das Zipfsche Gesetz (genauer Zipf's famous law, da G. K. Zipf diverse Gesetzmäßigkeiten von Sprachdaten untersuchte, vgl. Zipf 1935, Zipf 1949) herangezogen, um den Untersuchungsgegenstand auf seine Ähnlichkeiten zu natürlichsprachlichen Zeichensystemen zu überprüfen. Das Gesetz besagt, dass numerischer Rang und Häufigkeit von Einheiten natürlichsprachlicher Zeichensysteme insofern einen Zusammenhang haben, als dass ihr Produkt eine Konstante bildet. Um diese Konstante zu errechnen bzw. ihre Existenz zu überprüfen, muss man die Types[27] des Textes in eine Rangfolge gemäß ihrer Häufigkeit bringen. Der Type, der den höchsten Rang, also Rang 1 belegt, habe nun 1000 Vorkommen im gesamten Text. Der Type auf Rang 2 dürfte demnach 500 Vorkommen haben, der auf Rang 10 100 und der auf Rang

26 Eine zugegebenermaßen recht krude Chiffre. Weshalb Stallings näherliegende Chiffren, die den gleichen Effekt auslösen, nicht in Betracht zog (z.B. die aus Kapitel 5.3.2), ist unverständlich.

27 Type oder Typ wird von der analytischen Sprachphilosophie als allgemeiner Vorkommnistyp bezeichnet und steht im Gegensatz zum Token als einzelnem Vorkommnis.

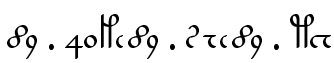
1000 dürfte nur einmal vorkommen, da $1000 * 1 = 500 * 2 = 100 * 10 = 1 * 1000$. Untersucht man natürlichsprachliche Texte, so sind die Produkte aus den mittleren Ränge und deren Häufigkeit tatsächlich annähernd konstant, starke Abweichungen finden sich dagegen bei den Produkten der vorderen (höhere Werte) und hinteren Ränge (niedrigere Werte).[28] Die von Zipf postulierte Worthäufigkeitsverteilung trifft die tatsächliche allerdings besser als eine Gleichverteilung. Mit diesem einfachen Kennwert können deshalb auch monoalphabetisch verschlüsselte Texte (die ihre Zipf-nahe Verteilung beibehalten) von polyalphabetisch verschlüsselten (die zu einer Gleichverteilung der Einheiten tendieren, siehe 5.2) unterschieden werden. Da der VM-Text im Groben der Zipf-Verteilung natürlicher Sprachen folgt (auf Zeichen- wie auch auf Wortebene), kann er nicht mit einer polyalphabetischen Chiffre verschlüsselt sein. Diese Verteilung wurde auch als Indiz dafür genommen, dass der VM-Text eine sinnvolle Nachricht transportieren müsste, da es nur mit großen Anstrengungen möglich sei, einen Text zu erzeugen, dessen Zipf-Verteilung der natürlicher Sprache entspricht, der aber keine natürliche Sprache ist, sondern nur sinnloses Gebrabbel.[29]

Eine sehr viel bemerkenswertere Eigenschaft des VM-Textes ist seine spezielle Wortlängenverteilung. Zwar entspricht die durchschnittliche Wortlänge mit fünf im ungefähren der mehrerer europäischer Sprachen, die Varianz ist dabei aber viel geringer (vgl. 6.1). Stolfi (2000) weist nach, dass sich die Wortlängenverteilung extrem der Binomialverteilung, basierend auf neun Zufallsexperimenten, angleicht. Er schließt daraus, dass VM-Wörter wesentlich mehr mit Zahlen zu tun haben, als mit Wörtern unserer Sprachen. Andere Forscher sehen in der Binomialverteilung einen Hinweis darauf, dass das VM auf einem Text beruht, der aus der Transkription einer Sprache des ostasiatischen Raums resultiert (vgl. 4.3.2).

Noch mehr Kopfzerbrechen bereiten den Forschern bestimmte Distributionseigenschaften der VM-Wörter. Es gibt Passagen, in denen das gleiche Wort mehrfach direkt hintereinander vorkommt, teilweise völlig identisch (aus Zeile 5 des ersten Paragraphen von f78r:  , transkribiert *qokedy qokedy dal qokedy qokedy*), teilweise mit leichten Abwandlungen (aus Zeile 2 des gleichen Paragraphen: 

28 Interessanterweise liefert das ursprünglich für die linguistische Anwendung konzipierte Gesetz für diverse andere Verteilungen sehr viel bessere Ergebnisse. Betrachtet man etwa die einwohnerstärksten deutschen Städte Deutschlands, so bleiben die Rang*Einwohnerzahl-Produkte relativ konstant (Einwohnerzahlen aus dem Jahr 2005, Quelle: Wikipedia): Berlin Rang 1 * Einwohner 3,40 Mio = 3,40; Hamburg 2*1,74 = 3,58; München 3*1,26 = 3,78; Köln 4*0,98 = 3,92 usw.

29 Inzwischen wurde allerdings von Li (1992) gezeigt, dass auch das willkürliche Tippen von Buchstaben Pseudowörter erzeugen kann, die ebenfalls der Zipf-Verteilung gehorchen.

 , transkribiert *qokeedy qokedy shedy tchedy.* Diese Abfolgen von Wörtern ähneln vielleicht niedergeschriebenen Flexionsparadigmen oder Übungen von Kindern, die das Schreiben lernen, aber sie ähneln nichts von dem, was gemeinhin unter Text verstanden wird.[30] Die hier angeführten Beispiele sind keine Einzelfälle, vielmehr gibt es dutzende solcher Vorkommen im Manuskript. Sie sind auch ein Beitrag zur sehr hohen Repetitivität des Textes.

Hinsichtlich der Wortentropie wartet der VM-Text mit einer weiteren Überraschung auf: Zwar weist er im Vergleich zu natürlichsprachlichen Texten völlig abweichende Zeichen-Entropiewerte auf, die Entropie für die Auftrittswahrscheinlichkeit für ganze Wörter aber liegt bei 10 Bit und ist damit natürlichen Sprachen sehr ähnlich. Wie diese offenkundige Diskrepanz erklärt werden kann, führt Zandbergen (2011c) aus: Die Auftrittswahrscheinlichkeit eines Wortes kann als Kettenfunktion für die bedingten Auftrittswahrscheinlichkeiten der einzelnen Zeichen des Wortes betrachtet werden. Für das Wort *so* etwa gilt (der Doppelpunkt steht für Wortanfang, der Unterstrich für Leerzeichen):

$$p(:\, so\_) = p(:\, s) * \frac{p(:\, so)}{p(:\, s)} * \frac{p(:\, so\_)}{p(:\, so)} \tag{4.1}$$

Auf dieser Basis ermittelt Zandbergen den Informationsgehalt der einzelnen Zeichen auf den verschiedenen Wortpositionen. Dabei stellt er fest, dass das Voynich'sche zwar sehr niedrige Werte für den jeweils ersten und zweiten Buchstaben der VM-Wörter hat, diese Positionen also sehr wenig Information tragen. Dafür holen VM-Wörter aber auf den restlichen Positionen bis zum Wortende auf, so dass sich summiert die gleiche Informationsdichte für VM-Wörter wie für die natürlicher Sprachen ergeben. Konkret untersucht Zandbergen Caesars *De Bello Gallico* (Wortentropie 10,16), die Vulgata-Genesis (9,33) und die Sektion mit kontinuierlichem Text des VM (Sterne-Sektion, 9,92). Die Information ist im VM sehr viel gleichmäßiger über die gesamten Wörter verteilt als in natürlichen Sprachen. Ein ähnliches Phänomen beobachtet Zandbergen bei der Analyse eines kunstsprachlichen Textes von Dalgano, was ein Hinweis darauf sein könnte, dass Friedman mit seiner Theorie, die er annagrammatisch verschlüsselt hinterließ, recht haben könnte (siehe Kapitel 4.3.2). Außerdem sieht Zandbergen durch seine Analyse bestätigt, dass VM-Wörter tatsächlich Wörter sind, da sie in ihrer Gesamtheit betrachtet variieren wie Wörter z.B. des Lateinischen.

30 Es sei aber bemerkt, dass der VM-Text der weiten Textdefinition aus Kapitel 2.2 ohne Zweifel entspricht.

Der innere Aufbau von Wörtern

Es wurde bereits thematisiert, dass sich adjazente VM-Wörter des öfteren sehr ähneln und sich evtl. nur durch den Wechsel einer einzigen Glyphe unterscheiden. Schon bei oberflächlicher Betrachtung des VM-Textes fällt auf, dass der Aufbau der Wörter bestimmten Ordnungsprinzipien folgt, auch wenn sie auf Anhieb nicht genauer spezifizierbar sind. Dieser Abschnitt widmet sich deshalb den bisherigen Ansätzen zur Aufdeckung der Regelmäßigkeiten, die den inneren Aufbau der VM-Wörter bestimmen.

Schon Tiltman (1967) entdeckte bei der Analyse der Sterne-Sektion, dass sich ein Großteil der Wörter in zwei Teile gliedern läßt, die er mit "Root" und "Suffix" bezeichnet. Jede Kombination eines Root- und eines Suffix-Elements ergibt dabei ein valides VM-Wort:

- **Roots**: ok/of – ot/op – qok/qof – qot/qop – ch – sh – d – s
- **Suffixe**: an/ain/aiin/aiiin – ar/air/aiir/aiiir – al/ail/aiil/aiiil – or – ol – ey/eey/eeey – edy/eedy/eeedy

Stolfi (1997a) erweitert das Tiltmannsche Schema, indem er die Wörter in Präfixe, Midfixe und Suffixe unterteilt. Seine Unterteilung basiert auf der Unterscheidung von *weichen* und *harten* Glyphen: Präfixe und Suffixe sind ausschließlich aus den weichen Glyphen aufgebaut (nach Stolfi *q, a, o, y, d, s, j, m, n, r, l, i*), Midfixe dagegen aus harten (Gallows, Ligaturen mit Gallows, *ch, sh* und *e*). Stolfi analysiert mit diesem Schema die Wörter der biologischen Sektion des VM. Bis auf sehr wenige Ausnahmen (kleiner 1%), bei denen weiche Glyphen innerhalb von Midfixen auftreten, passen alle Wörter in das Stolfi'sche Schema. Jedes der drei Elemente ist fakultativ, d.h. nach diesem Schema wohlgeformte Wörter können auf genau vier Arten aufgebaut sein: Präfix-Midfix-Suffix; Präfix-Midfix; Midfix-Suffix und Präfix-Suffix. Im letzten Fall lässt sich keine Grenze zwischen Prä- und Suffix erkennen (beide bestehen ja ausschließlich aus weichen Glyphen), weshalb sie Stolfi als Unifixe bezeichnet. Er stellt fest, dass es in der Klasse der Prä- und Suffixe nur sehr wenige Elemente gibt, die sehr häufig auftreten, während die Verteilung von Mid- und Unifixen gleichförmiger ist. Legt man die Annahme zugrunde, dass der Text einer Verschriftlichung einer nicht-indoeuropäischen Sprache entspricht (vgl.4.3.2), könnten VM-Wörter u.a. Silben darstellen. Stolfi schlägt vor, dass weiche Glyphen dabei Vokale, harte dagegen Konsonanten darstellen könnten. In einer weiteren Analyse (Stolfi 1999) testet der Autor verschiedene Möglichkeiten, Allographen (hier eigentlich Alloglyphen) und bestimmte Zeichen als Modifizierer zu detektieren. Auch dies tut er mithilfe einer internen Gliederung von VM-Wörtern, dem OKOKOKO-Schema, das er als mögliches

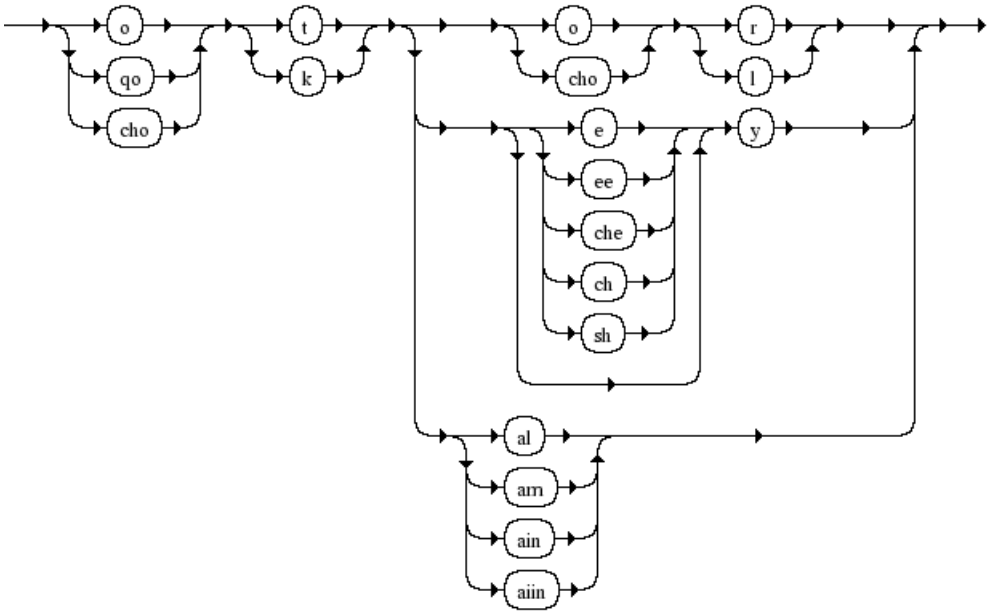


Abbildung 4.3: *Der innere Aufbau von VM-Wörtern nach Stolfi (1997a).*

Bauprinzip von VM-Wörtern annimmt. Die O-Positionen (ausgefüllt durch eines der Elemente {a, o, y}) sind dabei Modifizierer für die K-Elemente, die einer ungleich größeren Menge (Stolfi führt 59 auf) sogenannter Hauptelemente entstammen und wiederum eine dreiteilige Binnengliederung aufweisen.

Das Tiltmannsche Schema wurde auch von Roe[31] erweitert, indem er eine Grammatik für eine ganze Reihe von VM-Wörtern entwarf (Abbildung 4.3). Jeder Pfad dieses Graphen erzeugt ein Wort, das sich tatsächlich im VM-Text finden lässt.

Wie im weiteren Verlauf der Arbeit gezeigt wird, könnte der innere Aufbau der VM-Wörter wichtige Hinweise auf eine Entschlüsselung liefern. Später (in Kapitel 6) werden die hier aufgeführten Ansätze deswegen wieder aufgegriffen.

Dialekte und Syntax

Um die Aufzählung der statistischen Besonderheiten des VM-Textes abzuschließen, seien noch kurz die Thesen von Currier (1976) erwähnt, die sich einerseits mit der Verteilung des Vokabulars über verschiedene Seiten des VM, andererseits mit der Verteilung des Vokabulars innerhalb von Textzeilen beschäftigen.

31 Die Arbeit von Mike Roe selbst ist nicht mehr auffindbar, die Grammatik wird allerdings von Stolfi und Zandbergen auf deren Webseiten publiziert (Zandbergen 2011a).

Wie oben bereits geschildert, vermochte Currier auf verschiedenen Seiten des Manuskripts mindestens zwei unterschiedliche Handschriften zu detektieren, die distinkte statistische Eigenschaften hinsichtlich der enthaltenen Wörter aufwiesen. Daraufhin wurden u.a. von D'Imperio (1978a) diverse Clusteranalysen durchgeführt, die zwar zu unterschiedlichen Ergebnissen führten, aber durchsetzten, dass in der Folge von mindestens zwei Dialekten des Voynich'schen ausgegangen wird, die mit zwei verschiedenen Sprachen oder Verschlüsselungsmethoden erklärt werden könnten. In Dialekt A sind etwa 70% des Botanischen Abschnitts sowie die Pharmazeutische Sektion verfasst, Dialekt B wurde für den Rest der Botanischen sowie der Biologischen und der Sterne-Sektion verwendet. Der Astronomisch-Astrologische Teil kann nicht zweifelsfrei zugeordnet werden.

Bei der Betrachtung einzelner Zeilen des VM-Textes fiel Currier auf, dass Anfang und Ende dieser Textzeilen statistisch signifikante Merkmale aufweisen: Bestimmte Zeichen tauchen niemals am Anfang einer Zeile auf, andere dagegen sind fast ausschließlich im ersten Wort einer Zeile zu finden. Am Zeilenende finden sich dagegegen oft Zeichensequenzen, die nirgendwo anders im Manuskript auftauchen, was ein wenig den Eindruck erweckt, als wären die Zeilen um der Form willen mit bedeutungslosen Zeichen aufgefüllt worden. Currier sieht aus diesen Gründen Textzeilen des VM-Codes als funktionale Einheiten an, die zwischen der Wort- und der Paragraphenebene zu finden sind. In natürlichsprachlichen Texten bildet eine Zeile nur in Spezialfällen (etwa in der Lyrik) eine solche funktionale Einheit.

4.3 Kryptologische Untersuchungen

Obwohl sich die Voynich'sche Entdeckung schon bald zum 100. Mal jährt, ist es bisher nicht gelungen, eine Theorie zum Inhalt des Textes zu finden, die von der Mehrzahl der VM-Forscher anerkannt wird. Wie aus den in 4.1 erwähnten Korrespondenzen zu entnehmen ist, haben sich in der frühen Neuzeit zumindest in Prag mehrere Gelehrte um die Entschlüsselung des Textes bemüht, bevor er zu Kircher nach Rom kam und dort für fast 250 Jahre in verschiedenen Archiven verschwand. Über die Herangehensweise der Prager Fraktion um Baresch, Marci und Mnisovsky ist wenig bekannt, eine von der ersten Seite wieder entfernte Enschlüsselungstabelle von lateinischen und VM-Zeichen (vgl. Zandbergen 2011e) deutet darauf hin, dass einer von ihnen eine monoalphabetische Verschlüsselung (vgl. 5.1) angenommen hat. Monoalphabetische Verschlüsselungen sind allerdings sehr unsicher und ohne großen Aufwand zu brechen (vgl. 5.2), deshalb dürfte

eine solche dem Text kaum zugrundeliegen. Die Untersuchungen des 20. und frühen 21. Jahrhunderts gehen deswegen von anderen Möglichkeiten aus, wie der Text erzeugt worden sein könnte. Sie sind in drei Klassen einteilbar:

1. Der Text des VM ist ein mit einem Verschlüsselungsverfahren erzeugter Text.
2. Der Text des VM ist nicht verschlüsselt, sondern in einer evtl. unbekannten Sprache, jedenfalls in einem unbekanntem Schriftsystem niedergeschrieben.
3. Der Text des VM ist bedeutungslos, produziert als Gaunerei, um Geld zu verdienen.

Zumindest die Abgrenzung zwischen 1 und 2 ist unscharf. Nicht bekannte Schriftsysteme sind genau wie relativ wenig verbreitete Sprachen für kryptographische Zwecke eingesetzt worden (vgl. Kapitel 5.2), so dass man keine klare Trennlinie zwischen Texten ziehen kann, die unverständlich sind, weil sie verschlüsselt wurden, und Texten, die unverständlich sind, weil sie in einer unbekannten Sprache oder Schrift vorliegen. Um die Trennung aufrecht erhalten zu können, muss der Beweggrund des den Text erzeugenden Schreibers hinzugezogen werden: Unter der Herangehensweise 1 gruppieren wir alle Untersuchungen, bei denen die vorherrschende Intention des Verfassers die der Geheimhaltung war (4.3.1), unter Herangehensweise 2 werden alle Untersuchungen zusammengefasst, die davon ausgehen, dass der Verfasser nicht arglistig Klartext in Geheimtext überführt, sondern aus anderen Gründen die VM-Zeichen niedergeschrieben hat (4.3.2). Beiden Herangehensweisen gemeinsam ist, dass Sinn hinter den Zeichen vermutet wird, dass also mit ihrer Ausdrucksseite eine Inhaltsseite korrespondiert. Dies unterscheidet beide von der unter 3 erwähnten, die vor allem in den jüngsten Arbeiten zum VM vertreten wird (4.3.3).

4.3.1 Hypothese 1: Die Zeichen sind Chiffren

Unter der Annahme, dass die Zeichen Chiffren sind, entstanden die meisten Arbeiten zum VM; hier kann lediglich eine Auswahl dieser Arbeiten aufgeführt werden. In den Jahren nach der Entdeckung des VM besaß Newbold gleichsam exklusive Rechte an Studien zum Manuskript. Gemeinsam mit Voynich (und auch Rudolf II, wenn man dem oben erwähnten Brief an Kircher Glauben schenken darf) vertrat er die Meinung, Roger Bacon sei der Autor gewesen und dieser habe ein extrem kompliziertes, mehrstufiges Verfahren zur Verschlüsselung seiner Gedanken verwendet, das auf einer Mikroschrift basiert, welche um die mit bloßem Auge wahrnehmbaren VM-Glyphen verläuft. Newbold war sich sicher, eine wissenschaftshistorische Sensation enthüllt zu haben, da er durch seine Interpretation des Textes nachzuweisen glaubte, Bacon hätte bereits Teleskop und Mikroskop – lange vor

ihrer eigentlichen Entwicklung – zur Verfügung gehabt. Über Newbolds Annahmen ist im Detail schon oft berichtet worden,[32] so dass es hier wohl ausreicht, wenn erwähnt wird, dass sich mit der Verbreitung des Manuskripts nach dem Tode Newbolds durch seinen Schüler Kent die angenommene Mikroschrift als verlaufene Tinte entpuppte und die von Newbold postulierten Verschlüsselungen als Einwegverfahren klassifiziert wurden, so dass zwar eine Entschlüsselung, nicht aber die Verschlüsselung mit solchen Verfahren realisiert werden kann. Darüber hinaus bildete Newbold Anagramme aus über 50 Zeichen, so dass sich aus kombinatorischer Sicht eine nicht fassbare Anzahl unterschiedlicher Klartexte ergeben hätten. Jeder der Kritikpunkte für sich allein hätte ausgereicht, jede weitere ernsthafte Auseinandersetzung mit den Untersuchungen Newbolds zu diskreditieren. Ihr gesammeltes Auftreten hat dazu geführt, dass sich die über Jahre hinweg genarrte Mediävistik fast vollständig von der VM-Forschung verabschiedete.

Manly, der das Desaster, welches Newbold angerichtet hatte, als erster als solches benannte (Manly 1931), ging selbst von einer relativ einfachen Ersetzungschiffre mit einem exzessiven Einsatz von Nullen aus, äußerte sich aber nicht in einer Weise darüber, die von anderen Forschern hätte nachvollzogen werden können. Ähnlich liegt der Fall beim Krebsforscher Strong von der Universität Yale, der hinter der Chiffre ein "double reversed system of arithmetic progression of a multiple alphabet" (D'Imperio 1978b:36), also ein sehr kompliziertes polyalphabetisches Verfahren vermutete, dessen Funktionsweise er allerdings niemandem nahebringen konnte. Auch verdient der von ihm erzeugte Klartext seinen Namen nicht.[33] Feely (1943) präsentiert im Vergleich zu Newbold, Manly und Strong ein nachvollziehbares Verschlüsselungsverfahren, das auf lateinischen Abkürzungen beruht, scheitert aber wie Strong daran, damit einen akzeptablen Klartext zu generieren.

Diese Unzulänglichkeiten der kryptoanalytischen Angriffe, dass sie nämlich entweder auf nicht umzusetzenden Verfahren beruhten oder völlig inakzeptable Klartexte generierten (oder beides), führten dazu, dass die VM-Forschung, so sie denn überhaupt noch betrieben

32 Die Entschlüsselung des Textes beruht nach Newbold (Newbold & Kent 1928) auf einem insgesamt sechsstufigem Verfahren. Dabei musste zuerst die lateinische Mikro-Kurzschrift identifiziert und in der richtigen Reihenfolge angeordnet werden. Die einzelnen Buchstaben wurden teilweise verdoppelt, woraus sich neue Buchstabenpaare ergaben, die nach bestimmten Regeln gegen andere Paare ausgetauscht wurden. Für die Austauschpaare wurden die alphabetischen Werte anhand einer Ersetzungstabelle ermittelt, die in phonetische Werte und per Neuanordnung in einen sinnvollen Text überführt wurden (vgl. D'Imperio 1978b:34). Eine ausführliche und lesenswerte Beschreibung der Newbold'schen Theorie und ihrer Liquidierung findet sich in Kennedy & Churchill (2004:35ff).

33 Zur Verdeutlichung sei hier ein Ausschnitt aus Strongs "Klartext" aufgeführt: "When skuge of tun'e -bag rip, seo uogon kum sli of se mosure-issued ped-stans skubent, stokked kimbo-elbow crawknot." Strong (1945).

wurde, von anderen Voraussetzungen ausging: Man verabschiedete sich von der These, dem Text läge eine Chiffre zugrunde und begab sich auf ein anderes Terrain.

4.3.2 Hypothese 2: Die Zeichen entstammen einem unbekanntem Schriftsystem

Das spektakuläre posthume Scheitern Newbolds und die eher unbefriedigen Alternativtheorien von Strong, Feely und Manly, die in der unmittelbaren Folgezeit veröffentlicht und verworfen wurden, bewog einen der bedeutensten Kryptoanalytiker des 20. Jahrhunderts, William F. Friedman, sich mit dem VM auseinanderzusetzen. Gegen Ende des zweiten Weltkriegs waren eine Reihe von Experten[34] in der US-amerikanischen Hauptstadt Washington versammelt, die für die kryptoanalytische Kriegsführung eingesetzt wurden. Friedman formt mit ihnen die First Study Group (FSG, 1944-46), die das VM einer gründlichen Analyse unterzieht. Dazu konnten auch die ersten Computer genutzt werden, der Text wurde in diesem Zuge zum ersten Mal in eine computerlesbare Form transkribiert. Der hohe Aufwand, der durch diese Transkription entstand und die Tatsache, dass die Computertechnik erst in den Kinderschuhen steckte, führten dazu, dass vor der Demobilisierung der Gruppe durch die amerikanische Regierung nach Kriegsende nicht die gewünschten Ergebnisse produziert werden konnten. Friedman ließ aber schon nach kurzer Zeit verlauten, er halte es für sehr wahrscheinlich, dass der VM-Text auf einem synthetischen Sprachentwurf beruhe. Als die Computertechnik weiter fortgeschritten war, beruft Friedman eine weitere, die Second Study Group (SSG, 1962-63) ein, um diese Hypothese zu überprüfen. Der SSG aber sollte es ähnlich wie der FSG ergehen: Nachdem ein paar wenige computationelle Angriffe auf den VM-Text vorbereitet und teilweise schon durchgeführt waren, wurden die computertechnischen Ressourcen nicht weiter zur Verfügung gestellt (vgl. D'Imperio 1978b:42), so dass auch die SSG ihre Arbeit ohne ein endgültiges Ergebnis einstellen musste.

Was von den beiden groß angelegten und unglücklich gescheiterten Entschlüsselungsversuchen durch die Gruppen um Friedman bleibt, ist einerseits die erste vollständige Transkription des VM-Textes (FSG-Transkription, vgl. 4.2.2), andererseits die Idee, beim VM-Text könnte es sich um das Produkt einer Kunstsprache handeln. Wie es sich für einen Kryptologen gehört, hinterließ Friedman seine Hypothese nicht in Klartext, sondern ver-

34 D'Imperio (1978b:40) zitiert die ebenfalls zur Gruppe gehörende Kryptoanalytikerin (und Ehefrau von William F. Friedman), Elizebeth Friedman, dass dort "specialists in philology, paleography, ancient, classical, and medieval languages; Egyptologists, mathematicians, and authorities on other sciences depicted in the manuscript" zusammen arbeiteten.

schlüsselte sie durch ein Anagramm[35]: "I put no trust in anagrammic acrostic cyphers, for they are of little real value – a waste – and may prove nothing -finis". Es gibt allerlei – teils sehr amüsante – Lösungen für dieses Anagramm (Kennedy & Churchill 2004:141ff), erst nach Friedmans Tod wurde es final aufgelöst: "The Voynich MSS was an early attempt to construct an artificial or universal language of the a priori type. – Friedman."

Der Kryptoanalytiker Tiltman, der von Friedman in den frühen 1950er Jahren auf das Manuskript aufmerksam gemacht wird, verfolgt ebenfalls die These, dass es sich beim Text des VM nicht um eine Chiffre, sondern um eine künstliche Sprache und damit eine Art von Code handelt.[36] Tatsächlich, wie bereits in 4.2.3 erörtert, lassen sich hinsichtlich Wortaufbau und Entropiewerten große Ähnlichkeiten zwischen dem VM-Text und Kunstsprachen des 17. Jahrhunderts ausmachen. Tiltman (1967) nennt in diesem Zusammenhang die von Dalgano (1661) und Wilkins (1668) vorgeschlagenen artifiziellen Sprachkonstrukte.

Beiden von Tiltman ins Spiel gebrachten Kunstsprachen ist gemein, dass sie *Systeme a priori* sind, d.h. in allen Teilen konstruierte Sprachneuschöpfungen, die keinerlei Verwandschaft mit natürlichen Sprachen aufweisen.[37] Stattdessen waren sie philosophisch inspiriert; in der Sprache sollte die natürliche Ordnung der Dinge widergespiegelt werden. Primär für den Sprachentwurf war daher die Klassifikation grundlegender Begriffsvorstellungen oder Grundideen. Dalganos System beruhte auf einer eingeschränkten, etwa 1000 Elemente umfassenden Menge von elementaren Begriffen, aus denen komplexe Begriffe zusammengesetzt werden konnten. Wilkins' System dagegen war eine durchgehende logische Klassifikation. Diese Arbeit beschreibt er in seinem *Essay Towards a Real Character* (1668): Zunächst ordnet er die Dinge der Welt in hunderte von Tabellen ein (I), dann entwirft er eine *Natural Grammar*, die sowohl philosophisch, natürlich, als auch allgemein sein soll (II), um schließlich (I) in (II) umzusetzen. Er erfand dafür eine Begriffschrift (genannt *Real Character*), die aus von ihm erdachten Zeichen bestand. Dabei drücken ähnliche Zeichen ähnliche Konzepte aus, Grundkonzepte bestehen aus wenigen Strichen, Untergattungen fügen bestimmte Striche hinzu, mittels Ableitungs- und Flexionspartikeln können Derivative und Flexive gebildet werden. Wilkins scheint daran gelegen zu haben, den Unterschied zwischen Signifiant und Signifié zu beseitigen.[38]

35 Welches er darüber hinaus auch noch in einer Fußnote versteckte – Friedman & Friedman (1959:19).

36 Definitionen der Begriffe Chiffre und Code finden sich in 5.1.

37 Das unterscheidet sie von *Systemen a posteriori*, die natürliche Sprachen zum Vorbild haben, etwa *Esperanto* (Zamenhof 1887). Die Unterscheidung geht zurück auf Couturat & Leau (1903); die Autoren lassen auch Mischsysteme zu, etwa das *Volapük* von Schleyer (1880, vgl. Kniele 1889).

38 Es ist zwar nicht bekannt, dass sich wirklich jemand in der von Wilkins entworfenen Universalsprache unterhalten hat, aber die Beseitigung der Trennung von Signifiant und Signifié dürfte im praktischen

Läge dem VM tatsächlich eine Kunstsprache wie die beschriebenen von Wilkins und Dalgano zugrunde, könnten einige Phänomene des VM-Textes damit erklärt werden: Sowohl die sehr geringe Anzahl verschiedener Wörter als auch deren Ähnlichkeit zueinander könnten auf ein Sprachsystem a priori hindeuten. Nicht erklärt werden kann jedoch, wie es möglich ist, dass das gleiche Wort bis zu dreimal direkt aufeinander folgen kann. Friedman sieht die Sprachentwürfe auch als zu systematisch an, als dass sie auf die VM-Chiffre passen würden. Ein gewichtiges Argument gegen die Universalsprachentheorie ist auch, dass sie zeitlich nicht infrage kommt: Das VM war bereits in Prag bekannt, mehrere Jahrzehnte bevor die Sprachen von Wilkins und Dalgano entwickelt wurden. Tiltman müht sich vergeblich, einen Beweis für die frühere Invention einer synthetischen Sprache a priori zu finden. Was er dabei womöglich übersehen hat, ist die Tatsache, dass es schon weit früher mathematisch-kombinatorische Sprachentwürfe gab, die von der Kryptographie inspiriert waren und ihrerseits die späteren philosophischen Sprachentwürfe inspirierten (vgl. Strasser 1988b). Das heißt, es gab spätestens zu Beginn des 16. Jahrhunderts etwas, das wie eine *Kunstsprache aussah*, in Wirklichkeit aber eine *Chiffre war*. Wir werden später (im Kapitel 5.3.2) ausführlich darauf eingehen.

Jenseits der Theorie, der VM-Text sei das Produkt einer synthetischen Sprache a priori, werden weitere Ansätze verfolgt, die annehmen, dass es sich nicht um einen verschlüsselten, sondern um einen schlichtweg unbekannten Text handelt. Zandbergen etwa hält es für durchaus denkbar, dass der Text durch die Transkription eines Schreibers zustande kam, der die ursprünglichen Sprache (nach Zandbergen kämen Arabisch, Sanskrit oder andere Sprachen infrage), also das, was er transkribierte, nicht verstand (Kennedy & Churchill 2004:265). Relativ viele Anhänger hat die Theorie, der VM-Text der sei Klartext einer ostasiatischen/chinesischen Sprache, der in einer vom Schreiber selbst erdachten Buchstabenschrift verfasst wurde. Meist wird Guy als erster Vertreter dieser These genannt, Stolfi hat sie in zwei seiner Artikel aufgegriffen (Stolfi 1997b, Stolfi 2002) und sieht die Ergebnisse seiner Studien zur VM-Wortform (4.2.3) mit dieser Theorie konform. Die aktuelleste Arbeit (von Zbigniew Banssik), welche die ostasiatische Theorie vertritt, nimmt den VM-Text als Manchu-Klartext an, der Verfasser hat dafür ein eigenes Transkriptionsalphabet entworfen.[39] Gegen die Chinesisch-Theorie wird eingewendet, dass sich weder

Gebrauch Schwierigkeiten mit sich bringen: Sprachsignale werden nie sauber übertragen, so dass es oftmals Schwierigkeiten gibt, ein Signifiant zweifelsfrei zu erkennen. In einer Sprache, in denen fast gleiche Signifiants völlig unterschiedlichen Signifiés zugeordnet sind (wie in natürlichen Sprachen), dürfte die Disambiguierung über den Kontext sehr viel einfacher sein als in Sprachen, in denen ähnliche Signifiants für sehr ähnliche Konzepte stehen (wie in den beschriebenen Kunstsprachen).

39 vgl. `http://www.ic.unicamp.br/~stolfi/voynich/04-05-20-manchu-theo/`, zuletzt aufgerufen

im Text noch in den Abbildungen irgendein Anhaltspunkt finden lässt, der auf einen ostasiatischen Ursprung hindeutet. Identisches lässt sich einwenden gegenüber anderen als dem VM zugrundeliegend gedachten Sprachen wie Ukrainisch, Kreol-Flämisch und Hawaiianisch.

4.3.3 Hypothese 3: Den Zeichen fehlt die Inhaltsseite

Haben Zeichen keinen Inhalt, so sind sie semiotisch leer, es kann durch sie keine Information übertragen werden. Diese dritte Hypothese geht also davon aus, dass der Text des VM keinen Sinn ergibt, sondern ohne den Anspruch, eine Botschaft weiterzugeben – womöglich in der Hoffnung, einen finanziellen Gewinn zu erzielen – niedergeschrieben wurde. Mit anderen Worten: Es handelt sich um eine Fälschung (englisch *Hoax*, im weiteren werden wir die hierunter fallenden Ansätze als *Hoax-Hypothesen* bezeichnen). Wie schon im Kapitel zur Geschichte des VM (4.1) aufgezeigt wurde, besteht auch die Möglichkeit, dass Voynich das Manuskript nicht gefunden, sondern selbst angefertigt hat. Obschon die sehr einseitig geführte Korrespondenz zwischen Baresch und Marci in Prag auf der einen und Kircher in Rom auf der anderen Seite dagegen spricht, plädiert bspw. Kennedy dafür, dass Voynich die Fälschung nur besonders raffiniert angelegt haben könnte (vgl. Kennedy & Churchill 2004:276ff). Als hauptverdächtig aber wird von den meisten VM-Forschern weiterhin das Renaissance-Gelehrten- bzw. Scharlatan-Duo John Dee und Edward Kelley angesehen. Rational betrachtet aber könnte jeder, der seit dem Mittelalter des Schreibens mächtig war und Zugang zur damals sehr kostbaren Ressource Pergament hatte, und für den sich auch nur irgendein Beweggrund ausmachen lassen könnte, ein potentieller Erschaffer eines sinnfreien Dokumentes gewesen sein.

Die Hoax-Hypothese wird vor allem von den jüngsten in den kryptologischen Fachzeitschriften veröffentlichten Arbeiten zum VM gestützt. Ausgangspunkt sind immer die statistischen Besonderheiten des Manuskripts, die bisher jedem Entschlüsselungsversuch trotzten: „Previous research has failed to produce a plausible mechanism for generating substantial bodies of text with this features“ (Rugg 2004:31). Vor allem die hohe Repetitivität des Textes, die sowohl aus der inneren Struktur der Wörter, als auch in ihrer Ähnlichkeit untereinander genährt wird, wartet bis zum heutigen Tag auf ein Verschlüsselungsverfahren, das sie nachahmen könnte.

Ein aktueller Ansatz, diese Repetitivität im Rahmen einer Hoax-Hypothese zu erklä-

am 11.10.2011

ren, stammt von Rugg (2004). Auch er verdächtigt das Paar Dee/Kelley der Autorschaft und geht von den ihnen zur Verfügung stehenden Mitteln aus. Dee und Kelley gelten als Schöpfer der Kunstsprache *Henochisch*, die auch heute noch große Aufmerksamkeit in esoterischen Kreisen genießt. Henochisch war Dee zufolge die Sprache der Engel, mit der diese die Weisheiten Gottes dem Menschen vermitteln konnten. In mehreren Séancen konnten Dee und Kelley unter Zuhilfenahme einer Kristallkugel und mehrerer 49*49 Felder großer Tafeln von Buchstaben mehrere Dutzend Rufe der Engel (auf henochisch) inklusive ihrer Übersetzungen aufzeichnen. Rugg stellt nun die Annahme auf, die erwähnten Tafeln (die Kristallkugel benötigt er für seine Erklärung offenbar nicht) könnten, wären sie mit Silben[40] statt mit Buchstaben gefüllt, zur Konstruktion des VM-Textes führen. Die unterschiedliche Struktur von henochischen und voynich'schen Wörtern[41] veranlasst Rugg dazu, von der Annahme auszugehen, dass zur Erzeugung des VM noch zusätzlich ein von dem italienischen Mathematiker Jérôme Cardan um das Jahr 1550 entworfenes Cardan-Gitter (vgl. 5.2) genutzt wurde. Dieses könnte, auf diese Silbentabelle angewendet, einen VM-artigen Text erzeugen. Rugg entwirft eine solche Silbentabelle und behauptet, mit einem derartigen Verfahren in der Lage zu sein, einen Text im Umfang des VM in drei Monaten erzeugen zu können (was in etwa der Produktion von einer Seite pro Tag entspricht). Da aber längst nicht jede Form, die mit seiner Tabelle erzeugt werden kann, auch im VM-Text vorkommt, benötigt er zusätzliche Restriktionen, welche die generative Kapazität seiner Methodik einschränken (u.a. Einfärbungen von Silben, bestimmte Farbkombinationen werden als Wörter ausgeschlossen, vgl. Rugg 2004:39). Gerade diese zusätzlichen Verfahrensregeln lassen daran zweifeln, ob es sich bei der von Rugg postulierten Methode um ein wirklich praktikables Verfahren handelt, das eigens dazu entwickelt worden sein soll, einen sinnlosen Text zu erzeugen. Gleichwohl wird die Idee, hinter den Wörtern des VM könnte eine Ersetzungstabelle stehen, eine sehr fruchtbare, wenn man die Möglichkeit in Betracht zieht, dass die Ersetzungseinheiten sehr wohl auch eine sinnvolle Botschaft transportieren könnten (vgl. Anhang A). Wir werden später darauf zurückkommen.

Wo Rugg ein Verfahren sucht, welches die Auffälligkeiten des VM-Textes mit einer effektiven, in der Renaissance zugänglichen Methode nachahmen kann, verlegt sich Schinner (2007) vor allem darauf, weitere statistische Abweichungen des VM-Textes im Vergleich zu natürlichsprachlichen Texten zu erfassen und im Rahmen der Hoax-Hypothese zu inter-

40 Rugg nutzt dabei die von Stolfi ermittelten Prä-, In- und Suffixe als Silben

41 Leider führt Rugg die genauen Strukturunterschiede nicht aus. Auf den ersten Blick scheinen VM-Wörter hinsichtlich der Buchstabenkombinatorik viel stärker eingeschränkt, daneben erscheinen henochische Wörter längenvariabler.

pretieren. Er hält den VM-Text für das Produkt eines stochastischen Algorithmus; unter Zuhilfenahme des Random-Walk-Modells kann er aufzeigen, dass das VM im Gegensatz zu natürlichen Sprachen weitreichende Korrelationen zwischen den Einheiten des Textes aufweist (das Modell und Schinners Ergebnisse werden ausführlich im Kapitel 6.1 dargelegt). Solche weitreichenden Korrelationen sind für natürlichsprachliche Texte untypisch, sie entsprechen eher einer Zufallsauswahl, bei der sich nach einer bestimmten Anzahl von Zügen die Anfangsbedingungen wiederholen. Schinner sieht darin einen klaren Hinweis darauf, dass die Hypothese unter 4.3.2 zurückgewiesen werden muss, das VM also keinen Klartext, sei es Chinesisch oder eine Kunstsprache, enthalten kann. Stattdessen sieht er durch seine Untersuchung die Rugg'sche Hoax-Annahme wahrscheinlicher werden. Es ist aber zu bedenken, ob der Effekt der zufälligen Auswahl nicht auch auf andere Art erklärt werden kann, wenn nämlich eine Verschlüsselung von Klartexteinheiten durch mehrere verschiedene Chiffren angenommen wird. Innerhalb des Kapitels 6.1 werden wir uns dieser Thematik annehmen.

4.4 Zusammenfassung und Überleitung

Bevor wir dazu übergehen, Tesla als virtuelles Labor für die Prozessierung des VM-Textes einzusetzen, fassen wir zunächst die von der bisherigen VM-Forschung ermittelten und in diesem Kapitel ausgeführten Eigenheiten des Textes in kompakter Form zusammen (NaSp steht in der Aufzählung für *natürliche Sprachen*):

1. Die Zeichen entsprechen keinem bekannten Alphabet.
2. Die Zeichen sind – für Chiffren unüblich – zu Wörtern und Paragraphen gruppiert.
3. Interpunktionszeichen sind nicht auszumachen.
4. Die Häufigkeitsverteilung der Zeichen entspricht der NaSp (polyalphabetische Verschlüsselung kommt damit nicht infrage).
5. Die Verbundentropie ist extrem niedrig (von den NaSp weisen lediglich polynesische Sprachen ähnliche Werte auf, haben aber sehr viel weniger Grapheme).
6. Die Wortlängen sind eher kurz und binomial verteilt (was nur einige NaSp aufweisen, z.B. Übertragungen des Chinesischen in einer Buchstabenschrift).
7. Die Zipf-Verteilung der Wörter entspricht der von NaSp (was gegen eine Nonsens-Sprache sprechen soll).
8. Die Information ist innerhalb der Wörter annähernd gleichverteilt (in NaSp tragen die ersten zwei Zeichen deutlich mehr Information als die weiteren).

9. Das Vokabular (die Anzahl der Types) ist sehr klein.
10. Bestimmte Zeichen kommen nur in bestimmten Regionen innerhalb von Wörtern vor.
11. Die Wörter haben morphologisch einen sehr regelmäßigen Aufbau.
12. Wörter unterscheiden sich oft nur durch einzelne Zeichen voneinander (erinnert an Wörter flektivischer Sprachen).
13. Gleiche oder ähnliche Wörter treten in direkter Nachbarschaft zueinander auf (dies wird fast in sämtlichen NaSp vermieden).
14. Verschiedene Seiten scheinen in unterschiedlichen Dialekten verfasst.
15. Zeilen bilden eine funktionale Einheit (in NaSp nur in speziellen Textsorten, etwa Lyrik).
16. Weitreichende Korrelationen deuten auf einen Zufalls-Auswahl-Prozess hin (für NaSp, eigentlich für jede bedeutungstragende Kommunikation absolut unüblich).

Tesla soll nun einerseits dazu eingesetzt werden, diese wichtigen statistischen Merkmale des Textes in einer nachvollziehbaren Form zu berechnen, zu dokumentieren und schließlich zu veröffentlichen. Darüber hinaus sollen Merkmale auch mit denen anderer Texte verglichen werden, dabei kommen Texte aus verschiedenen Sprachen, aber auch Ergebnisse unterschiedlicher Chiffrierungen infrage.

Durch die Arbeit von Schinner wurde die Klartexthypothese weitestgehend widerlegt. Rugg hat gezeigt, dass man einen bedeutungslosen Text mit relativ einfachen Mitteln reproduzieren kann, was ein Hinweis auf, aber noch kein Nachweis der Hoax-Hypothese ist. Wie alle Hypothesen wird sie nicht verifiziert, sondern allenfalls falsifiziert werden können. Letzteres wäre der Fall, wenn man eine Chiffriermethode nachweisen könnte, mit der ein Text produziert werden kann, der vergleichbare Merkmale wie der VM-Text aufweist. Dieses Verfahren sollte darüber hinaus in die Zeit passen, die für die Entstehung des VM angenommen wird. Das nächste Kapitel wird sich auf die Suche nach einem geeigneten Kandidaten begeben.

Yet it may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.

(E. A. Poe: A Few Words on Secret Writing)

Kapitel 5

Ansatzpunkt: Kryptographie der frühen Neuzeit

In diesem Kapitel fahnden wir nach einem Chiffrierverfahren, mit Hilfe dessen ein Text erzeugt werden kann, der dem Voynich-Manuskript hinsichtlich der (am Ende des letzten Kapitels aufgezählten) grundlegenden Eigenschaften ähnlich ist. Das gesuchte Verfahren sollte in die Zeit passen, die für die Erstellung des VM infrage kommt. Wie in Kapitel 4.1 erläutert, spricht die neueste zeitliche Verortung des VM für eine Entstehungszeit nach 1404 (C14-Analyse des Pergaments) und vor 1608 (De Tempenecs Zeichen auf der ersten Seite). Die Möglichkeit, dass es sich beim VM um einen Text Roger Bacons aus dem 13. oder um eine Fälschung Voynichs aus dem 20. Jahrhundert handelt, wird aus diesem Grund hier nicht weiter verfolgt.

Die wissenschaftliche Auseinandersetzung mit Verschlüsselungsverfahren fällt in das Gebiet der Kryptologie, deren wichtigste Begriffe in 5.1 eingeführt werden. Daran anschließend folgt ein kurzer Abriss über die Geschichte dieser Wissenschaft (5.2). Die 200 Jahre, innerhalb derer das VM mit hoher Wahrscheinlichkeit entstanden ist, fallen in Europa auf den Übergang zwischen Mittelalter und Neuzeit. Vor dieser Zeit gab es, soweit bekannt, nur sehr einfache Verschlüsselungsverfahren, weshalb die Methoden, die während des Mittelalter/Neuzeit-Übergangs entwickelt wurden, bei den nachfolgenden Betrachtungen im Mittelpunkt stehen. Als besonders innovativ bei der Entwicklung neuer Chiffrierverfahren erwies sich zu dieser Zeit der Benediktinermönch Johannes Trithemius, dessen Leben neben seinen mitunter sehr ausgefallenen Methoden in 5.3 vorgestellt wird. Dabei geht es sowohl um die Verfahren seiner beiden kryptographischen Hauptwerke, als auch um deren Angriffsflächen für die Kryptoanalyse. Zum Schluss werden die Ergebnisse in 5.4 zusammengefasst und es wird zu den konkreten Anwendungen kryptoanalytischer Methoden in Kapitel 6 übergeleitet.

5.1 Glossar zur Kryptologie

Kryptologie ist die Wissenschaft, die technische Verfahren zur Informationssicherheit behandelt.[1] Historisch wurden solche Verfahren vor allem für die Kommunikationssicherheit (englisch *confidentionality*) eingesetzt, mit der weiten Verbreitung vernetzter elektronischer Informationsspeicher hat sich ein weiterer Anwendnungsbereich zum Schutz der Privatsphäre (englisch *privacy*) ergeben. Zusätzliche Einsatzbereiche kryptologischer Methoden sind die Authentifizierung als Sicherung gegen Fälschung und Unterschiebung sowie die Archäo-Linguistik, die sich mit der Entzifferung vergessener Sprachen und Schriftsysteme befasst (z.B. mit der Entzifferung von Hieroglyphen, die auch am Beginn dieser Arbeit thematisiert wurde). Information, die offen, also nicht verborgen und nicht verschlüsselt ist, nennt man in der Kryptologie **Klartext** (engl. *plain text*). Verfahren der **Kryptographie** überführen diesen Klartext in einen **Geheimtext** (engl. *cipher text*). Die komplementäre Methode, also Überführung des Geheimtextes in Klartext, nennt man Entschlüsselung oder **Kryptoanalyse**. Dem Prozess der Überführung von Klar- zu Geheimtext und umgekehrt liegt eine Kombination aus **Verschlüsselungsverfahren** (dem Algorithmus der Überführung) und **Schlüssel** (der Information über die Anwendung des Verfahrens) zugrunde. In der modernen Kryptographie gilt der Grundsatz des **Kerckhoff'schen Prinzips**: Die Sicherheit eines Kryptosystems beruht auf der Geheimhaltung des Schlüssels, nicht etwa auf der Geheimhaltung des Verfahrens.[2]

Bei den Verschlüsselungsverfahren wird **Kryptographie im engeren Sinne** (vom griechischen *kryptos* für Verbergen) von der **Steganographie** (vom griechischen *steganos* für Verschleiern) unterschieden: Kryptograpische Verfahren verbergen die Information, indem sie den Text verschlüsseln; steganographische Verfahren dagegen tarnen darüber hinaus die pure Existenz einer geheimen Information, indem diese in einem Maße verborgen wird, dass sie überhaupt nicht als Information wahrgenommen wird. In diesem Zusammenhang kann man zwischen technischen und linguistischen Verfahren unterscheiden: **Technische steganographische Verfahren** bedienen sich aus dem reichhaltigen

1 Die Definitionen der Begriffe sind zum großen Teil aus dem Glossar von Kahn (1996:xv ff) übernommen. Eine sehr ausführliche Darstellung der Klassifikation von kryptologischen Verfahren, versehen mit einer Fülle anschaulicher Beispiele, findet sich in Bauer (2000:9ff).

2 Von Claude Shannon (dem "Vater der Informationstheorie und des Digitalen Zeitalters", vgl. `http://wikipedia.org/wiki/Claude_Shannon`, zuletzt aufgerufen am 11.10.2011), wurde dieses Prinzip mit *the enemy knows the system* paraphrasiert. Das kontroverse Prinzip wird als *security through obscurity* (zu deutsch etwa *Sicherheit durch Unklarheit*) bezeichet. Bisweilen werden auch derartige Systeme für die Computersicherheit angewendet, dann allerdings meist in Kombination mit geheimgehaltenen Schlüsseln.

Fundus an Geheimtinten (bereits Kinder wissen um die Fähigkeiten von Zitronensaft) oder Verstecken (doppelte Böden, hohle Schuhsohlen, temporär kahlgeschorene Köpfe, verschluckte Mikrofilme, modifizierte Bild- oder Audiodateien usw.). **Lingustische steganographische Tarnverfahren** lassen sich in zwei Klassen einteilen: In **Semagrammen** ist die Tarnung sichtbar, wenn auch kaum wahrzunehmen, etwa indem einzelne Zeichen durch Mikropunkte, Tiefstellung oder Absetzung im Schriftzug als Klartextzeichen markiert werden. Im Gegensatz dazu ist bei sogenannten **Open Code Geheimschriften** die Tarnung auch nicht durch genaue Betrachtung wahrnehmbar. Zu dieser Klasse gehören maskierte und verschleierte Geheimschriften. **Maskierte Geheimschriften** verpacken die geheimzuhaltende Information in unauffällige Texte, indem die wahre Bedeutung unverfänglicher Floskeln im Vorhinein festgelegt wird. Dies entspricht der Entwicklung einer Geheimsprache, wie etwa die abweichende Bedeutung von Formulierungen in Arbeitszeugnissen.[3] Zu den maskierten Geheimschriften wird auch die im Kapitel 5.3.2 vorgestellte Ave-Maria-Chiffre gezählt, ein modernes Beispiel findet sich beim Webdienst *Spammimic*[4], wo beliebige Klartexte im Gewand typischer Spam-Mails getarnt werden. **Verschleierte Geheimschriften** setzen nicht auf die aufwendige Erfindung einer Geheimsprache, die Information kann stattdessen zurückgewonnen werden, indem die Nachricht gefiltert wird, da nur bestimmte Bereiche der Nachricht relevant sind; der Rest ist aufgefüllt mit sogenannten **Nullen**, d.h. Geheimtexteinheiten, denen keine Klartexteinheiten entsprechen. Auch für diese verschleierten Geheimschriften gibt es zwei Unterklassen: Bei der Verwendung eines **Würfels** finden sich die signifikanten Einheiten an bestimmten vorgegebenen Positionen (beim Spezialfall Akrostichon etwa am Anfang eines Wortes oder einer Zeile). Dagegen benötigt man für die **Raster**-Methode eine Schablone (nach ihrem Erfinder auch Cardan-Gitter genannt), welche die nicht-signifikanten, also nicht zum Klartext gehörenden Einheiten verdeckt.

Kommen wir zurück zu den kryptographischen Methoden im engeren Sinne. In diesem Bereich lassen sich zwei unterschiedliche Verfahren beschreiben: Transposition und Substitution. Grundidee der **Transposition** ist die Umstellung der Klartexteinheiten, also eine Änderung ihrer Abfolge. Als Beispiele hierfür sind Anagramme[5] oder auch die

3 Zur abweichenden Bedeutung bestimmter Ausdrücke in Arbeitszeugnissen vgl. etwa `www.arbeitszeugnis.de`, zuletzt aufgerufen am 11.10.2011.

4 Der Webdienst ist zu finden unter `www.spammimic.com`, zuletzt aufgerufen am 11.10.2011.

5 Etwa *Beispiel – Peilsieb*. Die Bildung herkömmlicher Anagramme, für die es keine exakten Transpositionsvorschriften gibt (etwa welche Ausgangsposition in welcher Endposition resultieren muss) ist allerdings kein kryptologisches Verfahren im engeren Sinne. Das liegt daran, dass die Transposition nicht eindeutig rückführbar ist (Verstoß gegen das Prinzip der Injektivität, s.u.). Damit kann die

Fleißner-Schablone[6] zu nennen. Verfahren der **Substitution** dagegen beruhen auf der Ersetzung von Klartexteinheiten durch Geheimtexteinheiten, die entweder Codes oder Chiffren sein können. Ersetzt man linguistische Einheiten (wie Morpheme, Silben, Wörter, Phrasen oder gleich ganze Sätze), so bezeichnet man das Substitut als **Code**. Werden dagegen Einzelbuchstaben oder n-Gramme ersetzt, so bezeichnet man das Substitut als **Chiffre**.[7] Bei der Verwendung von Chiffren definiert man die zu ersetzenden Einheiten (**Klartextalphabet**) und die Ersetzungseinheiten (**Chiffrenalphabet**). Für chiffrenbasierte Substitutionssysteme gilt im Übrigen noch folgende Terminologie: Wird ein Klartext in Einzelzeichen zerlegt, nennt man das Verfahren **monographisch**, wenn Zeichenkombinationen substituiert werden, spricht man von **polygraphischen** Verfahren. Bestehen die Chiffren aus Einzelzeichen, so ist das Verfahren **monopartit**, in den anderen Fällen bi-, tripartit usw. Wird zur Verschlüsselung lediglich ein einziges Chiffrenalphabet verwendet, ist das Verfahren **monoalphabetisch**, **polyalphabetisch** ist es bei der Verwendung mehrerer Chiffrenalphabete.

Als Nullen werden, wie oben erwähnt, Chiffren bezeichnet, die nicht für eine Klartexteinheit stehen, **Homophone** sind unterschiedliche Chiffren, die derselben Klartexteinheit zugeordnet sind, **Polyphone** dagegen nennt man Chiffren, die gleichzeitig für unterschiedliche Klartexteinheiten stehen. Die letzteren Konzepte sind Grundlage für zwei weitere wichtige Prinzipien von Kryptosystemen: **Das Prinzip der Injektivität**, auch als Linkseindeutigkeit bezeichnet, verlangt, dass jede Chiffre eindeutig auf eine Klartexteinheit zurückgeführt werden kann. Infolge dessen sind keine Polyphone erlaubt. Die Befolgung dieses Prinzips stellt sicher, dass jedem Chiffretext ein eindeutiger Klartext zugeordnet werden kann. Ein sogenanntes **Shannon'sches Chiffrierschritt-System** geht noch über dieses Prinzip hinaus, da es sowohl links- als auch rechtseindeutig ist. In einem solchen Verschlüsselungssystem muss demnach jede Klartexteinheit auf immer dieselbe Geheimtexteinheit abgebildet werden, mithin sind weder Polyphone noch Homophone erlaubt.

Die Klassifikation der Kryptosysteme ist in Abbildung 5.1 graphisch dargestellt. Unter

Überführung in ein Anagramm allenfalls als ein Einweg-Verschlüsselungsverfahren angesehen werden.

6 Benannt nach dem österreichischen Oberst Fleißner von Wostrowitz. Im Vergleich zum Cardan-Gitter werden durch die Schablone keine Nullen verdeckt, vielmehr ist es durch die durchdachte Anordnung der Aussparungen möglich, durch Drehung der Schablone die Abfolge der Klartexteinheiten festzulegen. Das Verfahren wird u.a. in Jules Vernes Roman *Mathias Sandorf* demonstriert, vgl. `http://www.staff.uni-mainz.de/pommeren/Kryptologie99/Klassisch/1_Monoalph/Sandorf/Billett.html`, zuletzt aufgerufen am 11.10.2011

7 Diese strenge Art der Unterscheidung ist selbstredend nur auf Buchstabenschriften anwendbar.

den jeweiligen Verschlüsselungs-Klassen sind - grau hinterlegt - ausgewählte Instanzen als Beispiel für die jeweiligen Verfahren aufgeführt, die in der Tesla-Komponente Cryptographer implementiert wurden (vgl. Anhang C). Sofern diese im Text noch nicht erwähnt wurden, werden sie in den folgenden Abschnitten kurz charakterisiert.

5.2 Geschichte der Kryptologie

Die Geschichte der Kryptologie ist weitestgehend bestimmt durch den Kampf um die Vorherrschaft zwischen Kryptographen und Kryptoanalytikern. Sobald es dem einen Lager gelungen war, über das andere zu triumphieren, löste dies auf der anderen Seite einen Entwicklungsschub aus, der die Verhältnisse irgendwann umkehrte. Gegenwärtig haben die Verschlüsseler die Oberhand gewonnen und mit Verfahren wie DES und RSA (s.u.) vorläufig die Möglichkeit geschaffen, Dokumente mit einfachen Mitteln – es genügt eine simple Umrechenmaschine – so zu verschlüsseln, dass sie ohne Kenntnis des Schlüssels nicht in akzeptabler Zeit zu entziffern sind. Diese Verfahren stehen durch die Verbreitung solcher Recheneinheiten wie bspw. PCs inzwischen so gut wie jedem zur Verfügung, so dass kryptoanalytische Angriffe auf moderne verschlüsselte Dokumente nahezu sinnlos sind.[8] Nicht ohne Grund wurde für diese Arbeit ein Dokument gewählt, das auf jeden Fall vor dem Aufkommen dieser Verfahren erzeugt wurde.

Betrachten wir aber nun die geschichtliche Entwicklung von Kryptosystemen. Die älteste bekannte Methode zur Verschlüsselung von Informationen ist die *Skytale*, entwickelt vor mehr als 2500 Jahren in Sparta. Für dieses Verfahren benötigte man nichts weiter als einen Stab mit definierter Dicke, auf den ein Band aufgewickelt wird, welches dann nicht von links nach rechts, sondern von oben nach unten zu beschreiben ist. Abgewickelt ergibt sich auf dem Band eine von der Information abweichende Reihenfolge der Zeichen, es handelt sich also um ein Transpositionsverfahren. Zur Rekonstruktion des Klartextes muss man das Band einfach wieder auf einen Stab der definierten Dicke aufwickeln (siehe Abbildung 5.2).

In der Antike wurde auch – von einem der bekanntesten ihrer Vertreter – das erste Substitutionsverfahren, die sogenannte *Caesar-Verschlüsselung* entwickelt. Hierbei handelt es sich um ein *monographisches monopartites monoalphabetisches Verfahren*, das die

8 Dies trifft jedenfalls solange zu, wie die Mathematik keine heute noch nicht absehbaren Fortschritte macht oder es gelingt, für diese Probleme anwendbare Quantencomputer zu bauen (vgl. etwa Singh 1999:383ff).

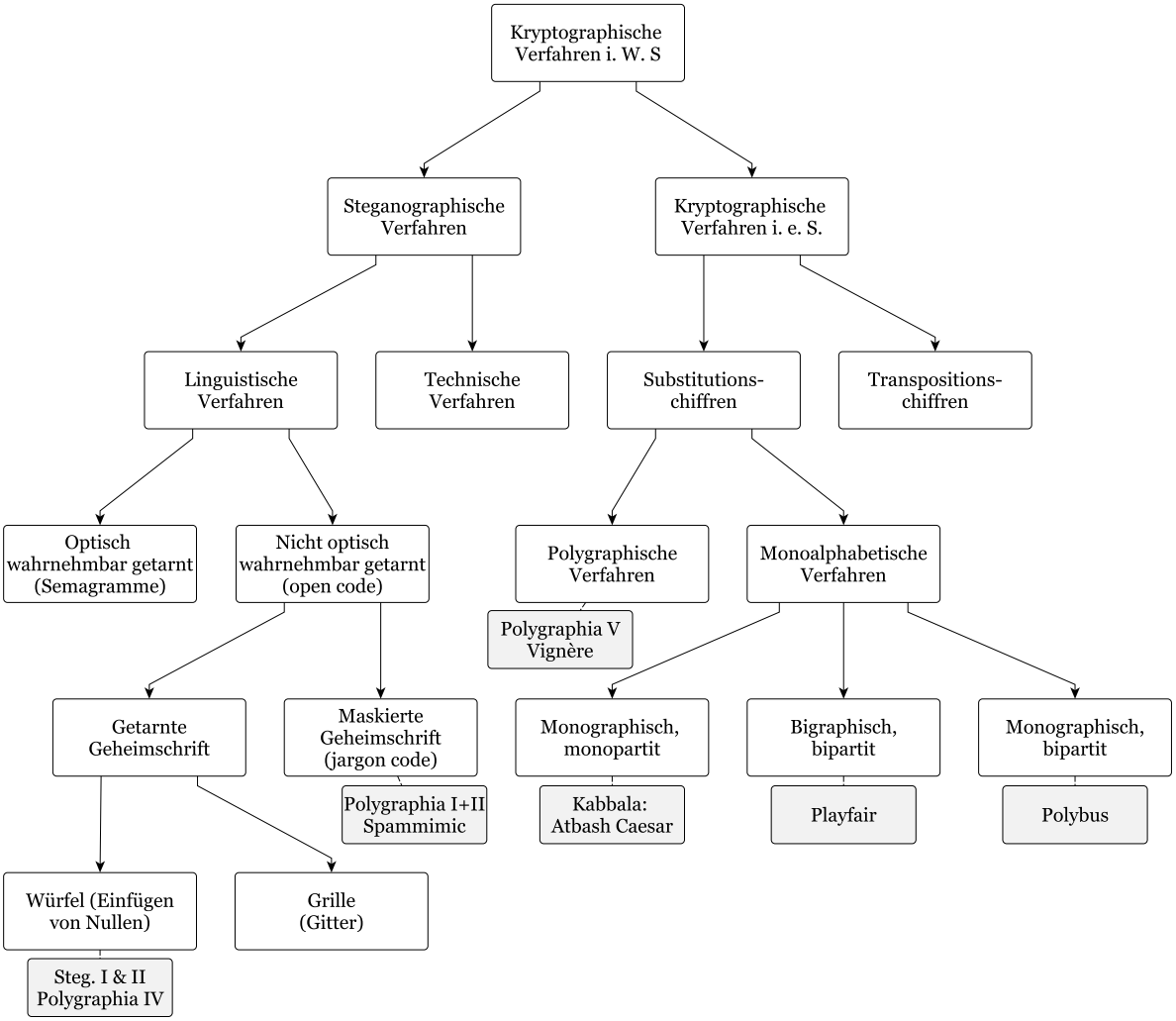


Abbildung 5.1: *Klassifikation kryptographischer Verfahren im weiteren Sinn. Grau hinterlegt sind die im Rahmen dieser Arbeit implementierten Verfahren.*

Abbildung 5.2: *Eine Skytale, bestehend aus einem Stab definierter Dicke und einem beschreibbaren Band. Quelle: http://commons.wikimedia.org/*

Verschlüsselung eines Textes erreicht, indem Geheim- und Klartextalphabet aus derselben Menge an Einheiten bestehen, die gegeneinander verschoben werden. Der Index des Caesar-Verfahrens bezeichnet dabei die Anzahl der verschobenen Positionen. In der Caesar-3-Chiffre etwa wird A, der erste Buchstabe des Alphabets, durch D, den (1+3=) vierten Buchstaben des Alphabetes substituiert, desgleichen B durch E usw. (vgl. Tabelle 5.1). Dieses Verfahren wurde bis ins hohe Mittelalter (in einigen Gegenden sogar noch länger) als sicher angesehen, obwohl seine Komplexität doch extrem eingeschränkt ist: Es existieren bei einem aus 26 Einheiten bestehenden Alphabet lediglich genausoviele Geheimtextalphabete. Ein *Brute-Force-Angriff*, d.h. das Ausprobieren aller Möglichkeiten, wäre hier mit sehr beschränkten Mitteln durchführbar und würde schnell zu einer Rekonstruktion des Klartextes führen.

a	b	c	d	e	f	g	h	i	j	k	l	m
D	E	F	G	H	I	J	K	L	M	N	O	P
n	o	p	q	r	s	t	u	v	w	x	y	z
Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Tabelle 5.1: *Die Caesar-3-Chiffre: Das Geheimtextalphabet (Großbuchstaben) ist um 3 Positionen gegenüber dem Klartextalphabet (Kleinbuchstaben) verschoben. Der Klartext* voynich *wird mit dieser Chiffre verschlüsselt zu* YRBQLFK.

Eine etwas höhere Sicherheit ließe sich dadurch erreichen, dass man ein durcheinander gewürfeltes Geheimtextalphabet benutzt, also etwa vereinbart, dass A für X steht, X für B, G für T etc. Insgesamt stehen mit dieser Methode N *Fakultät* mögliche Geheimtextalpha-

bete zur Verfügung,[9] die sich nicht mehr in akzeptabler Zeit nacheinander testen lassen. Allerdings lassen sich durch eine simple Häufigkeitsanalyse der Geheimtexteinheiten Kandidaten für korrespondierende Klartexteinheiten finden, zumindest, wenn die Sprache, in welcher der Klartext verfasst wurde, bekannt ist. In fast allen deutschen Texten ist bspw. das *E* mit Abstand der frequenteste Buchstabe und macht im Durchschnitt 17% eines Textes aus. Um die korrespondierende Geheimtexteinheit zu ermitteln, reicht es im Allgemeinen aus, diejenige auszuwählen, die am häufigsten auftritt.[10] Genauso verfährt man mit den dem E auf der Häufigkeitsskala folgenden Buchstaben *N, I, R, T* und *S*. Mit ein wenig Geduld und Einfallsreichtum lässt sich ein monoalphabetisch verschlüsselter Text so in relativ kurzer Zeit entziffern, was auch anhand der Beliebtheit von Zahlen-Kreuzworträtseln belegt werden kann. Das Verfahren wird Häufigkeitsanalyse genannt und wurde ursprünglich von arabischen Gelehrten entwickelt. Sehr langsam wurde das Wissen um diese Vorgehensweise der Kryptoanalyse von Süd- nach Nordeuropa weitergetragen, bis sich dann irgendwann der gesamte Kontinent davon überzeugt hatte, dass es um die Sicherheit monoalphabetischer Verfahren schlecht bestellt war.

Auf der Suche nach einem sichereren Verfahren, Informationen im Geheimen weiterzugeben, wurde zunächst die monoalphabetische Methode weiter modifiziert: die Einführung eines Nomenklators (einer Art Codebuch für die gebräuchlichsten Wendungen) erschwerte die auf Häufigkeitsanalyse basierende Decodierung genauso wie die Verwendung von Nullen, die erst einmal als Blendwerk erkannt werden müssen. Ganz unbrauchbar wird die Häufigkeitsanalyse, wenn man ein Geheimtextalphabet verwendet, in dem jeder Buchstabe eine Anzahl Homophone gemäß seiner Häufigkeit zugewiesen bekommt. So müsste etwa der Klartextbuchstabe *E*, der in deutschen Texten etwa 17% aller Buchstaben ausmacht, durch 17 verschiedene Geheimtexteinheiten dargestellt werden usw. Ein solches Chiffrieralphabet macht allerdings nur die auf einzelne Einheiten konzentrierte Häufigkeitsanalyse unmöglich. Wie die Kryptoanalytiker bald herausfanden, finden sich in natürlichen Sprachen auch Muster, die über die Häufigkeit von Einzelbuchstaben hinausgehen, so etwa die Häufigkeit von Bi- oder Trigrammen. Beispielsweise kann aus der Kenntnis heraus, dass *ER* und *EN* die mit Abstand häufigsten Bigramme im Deutschen ausmachen (ihr

9 Im Fall von 26 verschiedenen Buchstaben ergeben sich damit über 400.000.000.000.000.000.000.000.000 mögliche Geheimtextalphabete.

10 Die Methode funktioniert – wie eigentlich alle kryptoanalytischen Angriffe – umso besser, je länger die Chiffre ist. Große Schwierigkeiten würden wohl Leipogramme (Texte, in denen bestimmte Buchstaben nicht vorkommen) machen, bspw. Georges Perecs Roman *La Disparition* von 1969. Dieses Buch kommt, wie die deutsche Übersetzung unter dem Titel *Anton Voyls Fortgang*, gänzlich ohne den Buchstaben *E* aus.

Anteil an der Menge aller Bigramme macht zusammen über 20% aus), gezielt nach diesen Mustern gesucht werden. Homophone erschweren zwar die Identifizierung einzelner Buchstaben in einer monoalphabetischen Chiffre, durch n-Gramm-Analysen[11] ergibt sich aber dennoch ein großes Einfallstor für einen kryptoanalytischen Angriff.

Das Wissen um die Unsicherheit monoalphabetischer Verfahren führte in der Zeit des Übergangs vom späten Mittelalter zur Renaissance zu einer wahren Flut neuartiger kryptographischer Methoden. Die meisten von ihnen erwiesen sich allerdings als schwer handhabbar oder wurden schon nach kurzer Zeit von scharfsinnigen Kryptoanalytikern gebrochen, bis schließlich das polyalphabetische Verfahren entdeckt wurde. Bemerkenswert ist, dass die enorme Kapazität dieser Methode (immerhin wird sie noch heutzutage genutzt) offenbar nicht direkt erkannt wurde: Sie musste gleich dreifach – von Alberti im 15., von Trithemius Anfang des 16. und dann noch einmal von de Vigenère im 17. Jahrhundert – entdeckt werden, um sich schließlich durchzusetzen. Grundlage der polyalphabetischen Verschlüsselung ist – wie der Name schon andeutet – der Einsatz unterschiedlicher Ersetzungschiffren. Damit ist nicht die Verwendung von Homophonen gemeint, vielmehr werden Einheiten abhängig von ihrer Position in Klartext durch unterschiedliche Einheiten ersetzt. Um nicht immer neue Chiffren erfinden zu müssen, kann man der Einfachheit halber nur ein Ersetzungsalphabet verwenden, das aber nach jeder Klartexteinheit verschoben wird: Will man bspw. ein A auf der ersten Position ersetzen, wählt man ein B, ein A auf der zweiten Position aber wird durch ein C ersetzt usw. Das autorisierte Entschlüsseln des Textes wird damit natürlich auch sehr viel komplexer, da dem Empfänger die Information, welches Ersetzungssytem er an welcher Position anzuwenden hat, zukommen muss.

Die Verwendung polyalphabetischer Chiffren wurde erst mit Blaise de Vigenère wirklich handhabbar, der ein zu vereinbarendes Codewort einführte, welches den Schlüssel für die Verwendung der positionsabhängigen Klartextalphabete bildete (vgl. Tabelle 5.2). Mit diesem Verfahren, das einzig auf der Weitergabe eines Codeworts beruhte, welches sowohl Sender als auch Empfänger der Nachricht bekannt sein musste, hatten die Kryptographen vorerst die Oberhand gewonnen.

Das änderte sich Mitte des 19. Jahrhunderts, über 300 Jahre, nachdem de Vigenère seine Methode entwickelt hatte. Charles Babbage, der mit seiner *Analytical Engine* auch einen Vorläufer des modernen Computers entwarf, entdeckte 1854, dass eine Vigenère-Chiffre über Annahmen hinsichtlich der Schlüssellänge angreifbar war. Er veröffentlichte seine Ar-

11 N-Gramm steht für eine Verallgemeinerung über Bigramme, Trigramme usw.

Klartext	a	b	c	d	e	f	g	h	i	j	k	l	m	...
Reihe 1	**K**	L	M	N	O	P	Q	R	S	T	U	V	W	...
Reihe 2	**A**	B	C	D	E	F	G	H	I	J	K	L	M	...
Reihe 3	**H**	I	J	K	L	M	O	P	Q	R	S	T	U	...
Reihe 4	**N**	O	P	Q	R	S	T	U	V	W	X	X	Z	...

Tabelle 5.2: *Die Vigenère-Chiffre mit dem Schlüssel KAHN: Dem Klartextalphabet sind vier verschiedene Geheimtextalphabete zugeordnet, deren erste Ersetzungseinheiten die Buchstaben des Codewortes sind. Auf der ersten Position wird der Klartextbuchstabe durch die K-Reihe ersetzt, auf der zweiten durch die A-Reihe usw. Die fünfte Reihe entspricht wieder der ersten, die sechste der zweiten usf. Der Klartext "Voynich" wird mit dieser Methode verschlüsselt zu "FRFASCP". Man beachte, dass das erste F ein V, das zweite ein Y verschlüsselt!*

beit allerdings nicht, so dass die Methode unter dem Namen *Kasiski-Test* – nach Friedrich Wilhelm Kasiski (1668) – in der Kryptologiegeschichte zu finden ist. Dieser Test beruht auf der Wiederholung von Zeichenfolgen, die sich durch periodisch angewendete Schlüssel (wie es im Vigenère-Verfahren der Fall ist) und wiederkehrenden Mustern im Klartext zwangsläufig ergeben. Über eine Analyse des Abstands zwischen diesen Wiederholungen können Hypothesen über die Länge des Schlüssels aufgestellt werden. Eine polyalphabetische Chiffre, von der man die Schlüssellänge n weiß, ist aber nichts anderes als n aufeinanderfolgende monoalphabetische Verschlüsselungen. Je kürzer dabei der Schlüssel, desto einfacher die Dechiffrierung.

Damit wurde das Problem der Sicherheit polyalphabetischer Verfahren zu einem Problem der Schlüssellängen: Kurze Schlüssel waren einfach übermittelbar, aber mit dem Kasiski-Test unsicher geworden. Wenn hingegen der Schlüssel die gleiche Länge wie der zu verschlüsselnde Text hat, auf einer echten Zufallsverteilung basiert und nur ein einziges Mal genutzt wird, so spricht man von einem *One Time Pad*. Jeder kryptoanalytische Angriff auf ein mittels eines solchen Verfahrens verschlüsselten Text ist absolut sinnlos, weil sich jeder beliebige Klartext hinter der Chiffre verbergen könnte. Das Problem ist, dass das One Time Pad sowohl dem Sender als auch dem Empfänger zur Verfügung stehen *muss*, allen anderen aber nicht zur Verfügung stehen *darf*. Aufgrund dieser Verteilungsproblematik ist das Verfahren für den praktischen Einsatz im Allgemeinen nicht geeignet.

Erst durch Fortschritte der Mechanik bzw. des Maschinenbaus konnte sichergestellt werden, dass Sender wie Empfänger fast beliebig lange Schlüssel verwenden konnten: Die Zeit der Rotormaschinen war angebrochen. Eine solche Rotormaschine enthält mehrere bewegliche Walzen, die drehbar angeordnet sind und deren Stellung zueinander sich während

des Schlüsselvorgangs ändert. Damit ließen sich sehr lange Schlüssel aus kurzen generieren. Mit der bekannteste Verteter einer solchen Rotormaschine ist die *Enigma*, die im zweiten Weltkrieg von der deutschen Wehrmacht eingesetzt wurde. Dass sie trotz ihrer eigentlich ausreichenden Schlüssellänge (16900 Stellen) und ihres gewaltigen Schlüsselraums ($2*10^{20}$ Möglichkeiten) den kryptoanalytischen Angriffen nicht standhielt, lag teilweise an Konstruktionsfehlern, vor allem aber an unsachgemäßer Handhabung.[12]

Die echte Wende hin zur modernen Kryptologie vollzog sich dann mit der Einführung des Computers, der die Informationstechnologie revolutionierte. Vor allem sorgen die inzwischen fast flächendeckend verbreiteten Rechenmaschinen dafür, dass es für jedermann möglich ist, kryptologische Verfahren anzuwenden, ohne deren Arbeitsweisen nachvollziehen zu können. An dieser Stelle findet sich nicht genug Platz, angemessen auf die Entwicklungen der Informationssicherheit der letzten sechs Dekaden einzugehen; es sei nur kurz skizziert, auf welche Art moderne Verschlüsselungsverfahren in zwei Klassen geteilt werden:

- Bei **symmetrischen Verfahren** wird für Ver- und Entschlüsselung der gleiche Schlüssel genutzt (oder die beiden Schlüssel sind mit einfachen Mitteln ineinander überführbar). Der bekannteste Vertreter dieser Klasse ist der *Data Encryption Standard* (DES), der in den 1970er Jahren bei IBM entwickelt wurde. DES beruht auf einer Blockchiffre, in mehreren aufeinanderfolgenden Runden werden abwechselnd Permutations- und Substitutionsschritte angewendet. Dass die *National Security Agency* (NSA) Einfluss auf die endgültige Version von DES hatte, gab zu Spekulationen Anlass, diese Organisation hätte schon zum Zeitpunkt der Einreichung bei der Zertifizierungsagentur (1978) die Möglichkeit gehabt, die Chiffre zu brechen. Dessen ungeachtet erfuhr DES weite Verbreitung (z.B. in Geldautomaten), gilt aber inzwischen wegen der eingeschränkten Schlüssellänge von 56 Bit als unsicher. Bereits Ende der 1990er Jahren wurden DES-Schlüssel mehrfach durch Brute-Force-Angriffe gebrochen. Allerdings bedurfte es dafür noch eigens gebauter Maschinen. Um die Sicherheit zu erhöhen, wurden die verwandten Systeme *Triple-DES* und *AES* entwickelt.
- Bei **asymmetrischen Verfahren** gibt es einen Unterschied zwischen *öffentlichem* und *privatem* Schlüssel. Um einen geheimen Text an einen Empfänger zu schicken,

12 Die Geschichte der *Enigma* und ihrer Entzauberung durch Engländer und Polen wurde schon oft und sehr anschaulich beschrieben, vgl. etwa Kahn (1996:972ff) oder Singh (1999:179ff). Bauer (2000:195ff) fasst die Fehler, die im Umgang mit der Enigma gemacht wurden, auf kompakte Weise zusammen.

benötigt man dessen öffentlichen Schlüssel, mit dem man seine Chiffre erzeugt. Dieser Schlüssel kann allerdings nicht verwendet werden, um die Nachricht zu dekodieren, dafür benötigt man den privaten Schlüssel, der dem Empfänger exklusiv zur Verfügung steht. Dass es solche asymmetrischen Verfahren überhaupt geben kann, liegt an der Existenz *mathematischer Einwegfunktionen mit Falltür*. Diese Funktionen können in eine Richtung effizient berechnet werden (etwa die Multiplikation zweier großer Primzahlen), während dies für die entgegengesetzte Richtung nicht gilt (hier: Primfaktorzerlegung großer Zahlen), es sei denn, man hat bestimmte Zusatzinformationen (die Falltür). Es gibt allerdings keinen mathematischen Beweis dafür, dass es für Einwegfunktionen auf keinen Fall eine einfache Rückberechnung gibt, so dass sich die Verfahren irgendwann einmal als unsicher entpuppen könnten. Außerdem sind asymmetrische Verschlüsselungen (z.B. RSA, das auf der Primfaktorzerlegung beruht) im Vergleich zu symmetrischen sehr langsam. Da es aber ein großer Vorteil ist, dass der Akt des sicheren Schlüsselaustauschs entfallen kann, werden diese Verfahren oft angewendet, in der Praxis allerdings meist innerhalb hybrider Systeme, wie Pretty Good Privacy (PGP) von Phil Zimmerman aus dem Jahr 1991, in dem ein symmetrisches mit einem asymmetrischen Verfahren kombiniert wird.

Im kurzen Abriss der Geschichte der Kryptologie konnte nur auf die am weitesten verbreiteten Entwicklungen eingegangen werden, namentlich die einfache monoalphabetische sowie die Vigenère'sche polyalphabetische Methode, die später durch maschinelle bzw. computationelle Schlüsselgenerierung verbessert wurde. Hier sei zumindest erwähnt, dass es eine ganze Reihe von Verfahren gibt, die von diesen Schemata abweichen, so z.B. für die monoalphabetische Substitution die monographische bipartite Chiffre von Polybus sowie die bigraphische bipartite von Playfair.[13] Nicht betrachtet werden konnten zudem der reichhaltige Fundus an steganographischen Methoden (vgl. 5.1) und die Möglichkeit, unbekannte Sprachen für die Übermittlung von Geheiminformationen zu nutzen (vgl. Kahn 1996:549ff). Auch im Rahmen der jüdisch-mystischen Kabbala (vgl. 5.3.2) wurden eine Reihe kryptographischer Anweisungen ersonnen, die großen Einfluss auf die spätmittelalterlichen und frühneuzeitlichen Entwickler kryptographischer Verfahren hatten.

Bei der Suche nach einem Verfahren, das einen dem VM-Text vergleichbaren Geheimtext zu erzeugen vermag, wollten wir uns auf die Zeit zwischen dem Anfang des 14. und dem Ende des 15. Jahrhunderts einschränken. Tatsächlich wurde zu dieser Zeit auch der Grundpfeiler der modernen Kryptographie durch die (mehrfache) Entdeckung der polyalphabe-

13 Beide sind in der Kryptographie-Komponente implementiert (vgl. Anhang C).

tischen Methode gelegt. Der erste Entdecker war Leon Battista Alberti (1404-1472), jener "Uomo universale" (Burckhardt & Goetz 1922) und "Hochseiltänzer der Selbsterschaffung" (Grafton 2002), dessen kryptologische Tätigkeit neben seinen Erfolgen im Bereich der Architektur und der Kunsttheorie oft vergessen wird. Gleichwohl gilt er für Kahn (1996:130) als der unumstrittene Wegbereiter der modernen Kryptographie: "Alberti's three remarkable firsts – the earliest Western exposition of cryptoanalysis, the invention of polyalphabetic substitution, and the invention of enciphered code – make him the father of Western Cryptology". Albertis Verfahren sind aber nicht im mindesten so vielgestaltig wie die des deutschen Mönches Johannes Trithemius, der 10 Jahre vor Albertis Tod geboren wurde. Das kryptographische Werk dieses – im Verhältnis zu seinen Leistungen auf dem Gebiet – vergleichsweise wenig beachteten Kryptographen wird im Folgenden thematisiert.

5.3 Die Chiffren des Johannes Trithemius

Obschon Leon Battista Alberti der Titel „Vater der westlichen Kryptologie" gebührt, stehen in der vorliegenden Arbeit die kryptographischen Systeme des Johannes Trithemius im Fokus. Es scheint, als habe dieser völlig ohne konkretes Vorbild, allenfalls von diffusen im Mittelalter bekannten Methoden angeregt, neue und eigenständige Verfahren der Verschlüsselung von Texten entwickelt. Sein Einfluss auf ihm nachfolgende Theoretiker kryptographischer Systeme wie De Vigenère und Mnisowski ist vielfach belegt und kann fraglos als außerordentlich prägend für die Kryptographie der Neuzeit angesehen werden. Darüber hinaus pflegte er Umgang mit all den illustren Gestalten der frühneuzeitlichen europäischen Magie und Alchemie, die allesamt der Verfasserschaft des VM verdächtig erscheinen, auch er selbst wird „als Verfasser alchemistischer Traktate und anonymer sektiererischer Elaborate immer wieder in Anspruch genommen" (Arnold 1971:187).

Der Verlauf von Trithemius Leben nahm Einfluss auf die Entwicklung seiner Chiffren, genau wie die Rezeption seiner Chiffren Einfluss auf den Verlauf seines Lebens nahm. Einzeln betrachtet gelänge die Darstellung der von ihm entworfenen Verschlüsselungsverfahren deshalb nur lückenhaft. Aus diesem Grund ist der eingehenden Betrachtung dieser Verfahren (5.3.2) eine kurze Biographie vorangstellt (5.3.1). Am Ende dieses Kapitels erfolgt eine Bewertung der möglichen Angriffspunkte auf ausgewählte trithemische Chiffren (5.3.3).

5.3.1 Auftakt zur kryptographischen Neuzeit: Johannes Trithemius

Trithemius wurde am 1. Februar des Jahres 1462 in Trittenheim an der Mosel als Johannes Heidenberg, Sohn eines Winzers, geboren. Kurz darauf starb sein Vater, seine Mutter heiratete einige Jahre später erneut. Der Stiefvater, ein Gegner jeglicher schulischer Bildung, war dafür verantwortlich, dass der junge Johannes bis ins Alter von 15 Jahren Analphabet blieb, sich aber offensichtlich extrem nach Unterrichtung sehnte, wie er später selbst schildert:

> „Eines Nachts erschien ihm im Traum die hellgekleidete Gestalt eines Jünglings, der zwei Tafeln in den Händen hielt. Die eine war mit Schriftzeichen, die andere mit Bildern versehen. Aufgefordert zu wählen, nahm er, obgleich des Schreibens unkundig, 'amore litterarum' die beschriebene Tafel. Darauf sagte der Jüngling: 'Sieh, Gott hat deine Gebete erhört und wird dir beides zuteil werden lassen, das du erbeten hast, und zwar mehr als du erbitten konntest'“ (Arnold 1971:7, übersetzt aus der Chronikon Sponheimense, S.395)

Der starke Drang nach Wissen im jungen Mann ließ diesen schließlich von zu Hause ausreißen und auf eine Bildungsodyssee gehen, die ihn vermutlich über Trier, die Niederlande und Köln nach Heidelberg führte, wo er eine umfassende humanistische Ausbildung unter Reuchlin, Dalberg und Wimpfelig erhielt; ein universitärer Abschluss ist für ihn indes nicht nachgewiesen. Noch vor seinem 20. Geburtstag trat er in das Benediktinerkloster Sponheim bei Bad Kreuznach ein und legte im November des Jahres 1482 das Ordensgelübde ab. Schon kurz darauf, im Juli 1483, mit gerade 21 Jahren, wurde er zum Abt des Klosters gewählt. Er hat diese Position für die nächsten 25 Jahre ausgefüllt und dabei die weitaus bedeutednste Epoche in der Geschichte des Klosters begleitet (Arnold 1971:12). In dieser Zeit konsolidierte er die wirtschaftlichen Verhältnisse des Klosters und tat sich als eifriger Verfasser von Chroniken, literaturhistorischen Schriften und theologischen Werken hervor. Daneben stand er im Dienst der monastischen Reform, einer großen benediktinischen Ordensreform im 15. Jahrhundert. Vor allem aber kümmerte er sich um die Erweiterung der Bibliothek seines Klosters:

> “Schon nach kurzer Zeit hatte Trithemius auf der Pirsch die Klosterbibliotheken von Kreuznach, Mainz, Köln, Trier, Fulda, Weißburg, Frankenthal, Hirsau etc. durchstöbert. Den Mönchen vor Ort kaufte er interessante Bücher und Handschriften über Theologie, Astronomie, Philosophie, Musik, Rhetorik und Geschichte ab oder unterbreitete entsprechende Tauschangebote. Rasch eil-

> te Trithemius der Ruf voraus, ein echter 'bibliophagus', ein Bücherfresser zu sein." (Kuper 1998:37f)

Durch die von ihm forcierten Reformen sowohl an seinem eigenen Kloster als auch beim Orden im Allgemeinen, durch seinen Fleiß bei der Anfertigung der ausführlichen Sponheimer Klostergeschichte (*Chronicon Sponheimense* 1495-1506/9), der ersten deutschen Literaturgeschichte (*Catalogus illustrium virorum Germaniae* 1491-1495) und diverser theoretisch-theologischer Schriften, erarbeitete sich Trithemius einen exzellenten Ruf und Respekt unter seinen deutschen und europäischen Zeitgenossen. 1494 entfacht er dann urplötzlich und vermutlich völlig unbeabsichtigt einen Gelehrtenstreit, in dessen Folge er sich zum ersten Mal Anfeindungen gegenübersieht, die nicht nur inhaltliche Aspekte seiner Arbeit betreffen, sondern auch die Art seiner Recherche angriffen. Quell des Streites war eine Ausführung Trithemius' in seinem Buch *De laudibus sanctissimae matris Annae*, in der er sich als Anhänger der These offenbart, Maria wäre durch ihre Mutter Anna unbefleckt empfangen worden.[14] Trithemius sah sich einigen teils polemischen, teils mit beißender Ironie angereicherten Traktaten konfrontiert, die seinen Standpunkt in dieser Sache ins Lächerliche ziehen wollten. Die Tatsache, dass dieses Dogma bis zum heutigen Tage Teil der von der katholischen Kirche verbreiteten Lehre ist, macht schnell deutlich, zu wessen Gunsten dieser Streit ausging, was vor allem auch dem Eingriff von einflussreichen Freunden Trithemius' aus dem deutschen Humanistenkreis zu verdanken ist. Trithemius erweckte mit seinem Werk also auch Widersprüche. Aus diesem ersten Scharmützel ging er ohne Zweifel als Sieger hervor, anders verhält es sich bei der nächsten Episode, die unmittelbar mit seinem ersten kryptologischen Werk, der *Steganographia*, verbunden ist.

Nach dem Vorbild der symbolischen Kabbala (s. Abschnitt 7.3.1) verfasst Trithemius im letzten Jahr des 15. Jahrhunderts, gleichsam am Scheitel der Zeitenwende, ein Werk, welches als eine Anleitung zum Gebrauch von Geheimschriften dienen soll. Er ist sich dabei der Gefahren dieser Techniken sehr wohl bewusst[15] und will deshalb seine Ausführungen nur verantwortungsvollen Fürsten zugänglich gemacht sehen. Letztlich ist er aber nicht vorsichtig genug, oder einfach zu redselig: Er verfasst ein Schreiben, eine Art Vorankündigung zur Steganographia, das zu Missverständnissen geradezu herausfordert: Ein großes,

14 *Immaculata conceptio* – Dieses Dogma ist nicht zu verwechseln mit der jungfräulichen Geburt von Jesus durch Maria. In der christlichen Mythologie ist die Konstruktion offenbar notwendig, um Maria von der Erbsünde befreien zu können.

15 So erörtert er in der Vorrede zur Steganographia, welch großes Unheil leichter Zugang zu den im Buch folgenden Inhalten nach sich ziehen würde: Ehebruch, Hurerei, Rottierung, Verräterei, Raub, Mord und weiteres Unglück. Fielen die von ihm behandelten Methoden den Falschen zu, so "würde weder Treu noch Glaube auf der Welt mehr statt finden" (von Stieler 1673:495).

allseits Erstaunen auslösendes Werk wird angekündigt, unglaubliche Dinge werden vorgeführt, die trotz stetiger Beteuerung, alles würde auf natürliche Weise vollführt, Angst und Skepsis unter den Gottesgläubigen auslösen müssen.

> "Denn was wird nicht alles versprochen: Im ersten Buch eine Geheimschrift, die in einem unverdächtigen Text verborgen sein soll, im zweiten eine Technik der Nachrichtenübermittlung mit Hilfe eines Boten, der von der Botschaft selbst keine Kenntnis hat, oder auch ohne Zuhilfenahme eines Boten; das dritte Buch ermögliche es einem ungelehrten Manne, der nur seine Muttersprache beherrscht, innerhalb von zwei Stunden lateinische Briefe zu lesen, zu schreiben und zu verstehen, das vierte soll lehren, Nachrichten beim Essen, Sprechen oder Musizieren mitzuteilen. Der Inhalt des fünften Buches soll noch unbekannt bleiben, alles genannte jedoch ohne Täuschung oder magische Kunststücke vonstatten gehen." (Arnold 1971:182)

Trithemius Leben hätte vermutlich einen anderen Verlauf genommen, wäre die – zugegebenermaßen recht großspurige – Vorankündigung tatsächlich nur demjenigen zugekommen, für den sie auch bestimmt war, namentlich dem Genter Karmeliter Arnold Bostius.[16] Der aber verstarb bedauerlicherweise vor der Zustellung und so nahmen die Dinge einen für Trithemius verhängnisvollen Lauf: Der Brief wurde vom Prior der Genter Karmeliter geöffnet und sein Inhalt überall verbreitet. Das führte zu großem Interesse am noch sehr fragmentarischen Werk der Steganographia, unterlief Trithemius Vorsatz der Geheimhaltung und führte letztlich dazu, dass das Buch für ein Zauberbuch eines okkulten Schriftstellers gehalten wurde. Was geschah? Unter den Besuchern, welche das angekündigte Werk in Augenschein nehmen wollten, befand sich unter anderen auch der Pariser Philosoph und Theologe Carolus Bovillus. Zwar wich Trithemius von den von ihm selbst gesetzten Vorgaben der Vorankündigung beträchtlich ab, als er die Steganographia realisierte, die Geheimschriften des Werks dürften aber dennoch für Aufsehen gesorgt haben, da sie auf Beschwörungsformeln beruhten, die Trithemius von „Geisternamen" aus der Kaballa entlehnte (vgl. 5.3.2). Bovillus zeigt sich über den Text entsetzt und bezeichnet ihn als Gotteslästerung. „Nach seinem Urteil ist das ganze Werk nicht mehr wert, als den Flammen übergeben zu werden." (Arnold 1971:183). In der Folge wird der fromme Trithemius in den Augen der Öffentlichkeit zum okkulten Magier, ein Ruf, der auch nach seinem Tod an ihm haften bleibt. Gedruckt erscheint die Steganographia, auch auf Trithemius Wunsch hin, sie nicht zu veröffentlichen, erst 1606, also mit mehr als 100jähriger

16 Der Brief ist als Quelle in Anhang E aufgeführt.

Verspätung. Kurze Zeit später, im September 1609, landet sie auf dem Index der von der katholischen Kirche verbotenen Bücher und bleibt dort mehr als 250 Jahre.[17]

Über all die Vorwürfe und Anfeindungen, denen sich Trithemius jetzt – auch von den Mönchen seines eigenen Klosters – ausgesetzt sieht, gibt er den Vorsitz von Sponheim, das er einst zur Blüte geführt hat, auf, um Abt im Würzburger Schottenkloster zu werden (1506). Die von ihm zusammengetragene und weithin angesehene Bibliothek kann er trotz mehrfacher Bemühungen nicht mit überführen. Die Zeit im neuen Kloster beginnt dessen ungeachtet mit einer sehr arbeitsamen und kreativen Phase: Innerhalb der nächsten zwei Jahre vollendet Trithemius die *Chronicon Sponheimense* (1506, Nachträge 1509) und liefert eine nachträgliche Rechtfertigung für die Geisterbeschwörungsformeln in der *Steganographia* (*De septem secundeis*, 1508).[18] Daneben stellt er sein zweites kryptographisches Werk, die *Polygraphia*, fertig (siehe 5.3.2), die er 1508 anlässlich des Kölner Reichstages dem Kaiser des *Heiligen Römischen Reiches deutscher Nation*, Maximilian I. übergibt.

Obschon Trithemius Verdienste auf dem Gebiet der Kryptographie einen für die Nachwelt unbestrittenen Einfluss haben, stammen aus seiner Feder auch zumindest als fragwürdig zu bezeichnende Ausführungen. Er selbst ist offenbar der Existenz von Hexen gewiss und verfasst, basierend auf den Anschauungen des auch schon zu seiner Zeit umstrittenen Hexenhammers (*Malleus maleficarum* von 1487) eigene Abhandlungen über die schwarze Magie (*Antipalus maleficiorum* von 1508, als Auftragsarbeit für den Fürsten Joachim von Brandenburg, sowie das verschollene *De daemonibus*). Möglicherweise auch unter dem Druck, selbst als Magier verunglimpft zu werden,

> "hat der Magieexperte Trithemius unterschiedslos hermetisch-magische Traditionen in einen zum Auskochen bestimmten Topf geworfen, was man einem Kenner und Feinschmecker des elitären Schrifttums der antiken und mittelalterlichen Gelehrtenmagie nicht unbedingt zutrauen würde. In der trüben Suppe des Zaubereintopfes schwimmen bröckchenweise Ptolomäus, Geber, Hermes, Claviculae Salomonis neben den schillernden Arnold de Villanova und Peter von Abano." (Kuper 1998:107)

Nach Trithemius Anschauung darf man gotteslästerliche Bücher nicht verbrennen, sondern muss sie an einem sicheren Ort, wie bspw. einem Kloster, bewahren, um auf die Gefahren,

17 In dieser Zeit streitet die katholische Kirche ab, dass Trithemius das Werk tatsächlich verfasst hat.

18 Trithemius stellt in dieser Abhandlung klar, dass Gott den Kosmos regiere und sich dabei der sieben Planetengeister bediene, die jeweils eine bestimmte Zeit – 354 Jahre und 4 Monate – an der Macht bleiben.

die sie in sich tragen, angemessen reagieren zu können. Gläubige Christen schützten sich gegen schwarze Magie und gegen von Dämonen verursachte Krankheiten durch festen Glauben und die Vermeidung von Sünden. Hexen hingegen müssten exorziert werden, und zwar durch ein neun Tage währendes Dauerbad, das mit unterschiedlichsten Zusätzen versetzt ist. Die Erwähnung eines solchen Bades ist vielleicht der einzige Bezug, der sich zwischen Trithemius und den *Zeichnungen* des VM ausmachen lässt.

Findet man im Nachhinein vielleicht Erklärungen für diese eigentlich schon zu seiner Zeit überholten Vorstellungen, diskreditiert sich Trithemius durch seine Arbeit als Chronist der germanischen Stammesgeschichte endgültig: Getrieben durch die im deutschen Humanismus fußende Idee, die Germanen stammten direkt von den Griechen ab, im Speziellen lasse sich das Geschlecht der Habsburger auf Hektor von Troja zurückführen, erfindet er einen Geschichtsschreiber (Hunibald), dessen Chroniken er in seinen Arbeiten zitiert. Kaiser Maximilian, interessiert an diesem ihm unbekannten Geschichtsschreiber, verlangt, die Chroniken zu sehen, einen Wunsch, den Trithemius ihm aufgrund der Fiktionalität dieser Schriftstücke natürlich nicht erfüllen kann. Trithemius sieht sich immer mehr in die Enge getrieben, wird allerorts kritisiert und als Märchenerzähler verspottet. Am 13.12.1516 schließlich stirbt er und hinterlässt der Nachwelt ein gespaltenes Bild von sich:

> „Er liebt und liest die alten Kirchenväter und Schriftsteller des Mittelalters, ist aber gleichzeitig ein Freund und Förderer der humanistischen Studien, so dass er nicht nur das Lateinische treibt und schreibt, sondern sich auch dem Hebräischen und Griechischen, Byzantinischen zuwendet, und bereits für alte Denkmäler der Muttersprache Verständnis aufbringt. Er kennt und bekämpft die Schwächen der damaligen abendländischen Kirche, bleibt ihr indessen treu. Er verwirft Zauberei, Magie, Astrologie und Alchemie, kann sich trotzdem den Einflüssen der okkulten Wissenschaften nicht entziehen. Er behauptet und versucht ein wahrheitsliebender, gewissenhafter historischer und literaturkundlicher Autor zu werden und zeigt eine von seiner eigenen Bibliothek und auf vielen Bibliotheksreisen gespeiste außerordentliche Quellenkenntnis, aber ist sehr häufig unkritisch, irrt sich und ist flüchtig, ja er verfälscht, verwechselt und erfindet Texte. Seine Verstöße gegen die Wahrhaftigkeit haben dem Nachruhm des, wie gesagt, oft wahrheitsliebenden Schriftstellers außerordentlich geschadet.” (Lehmann 1961:4)

Und dennoch – Trithemius nahm großen Einfluss sowohl auf zeitgenössische, als auch nachfolgende Gelehrte, hier vor allem auf die, welche mit der Magie oder den Geheimwis-

senschaften in Verbindung stehen:

> "Drei Namen sind es vornehmlich, die mit der Geschichte der Magie zu Anfang des 16. Saeculums in Verbindung stehen: Dr. Faustus, Agrippa von Nettesheim und – wenn auch bedingt – Paracelsus. Mit jedem dieser drei ist Trithemius in Verbindung zu bringen, stärker noch hat die Nachwelt das Bild ihrer Persönlichkeit zur Deckung gebracht. Für die Existenz des historischen Faust ist der Abt die erste literarische Quelle." (Arnold 1971:185)

Obschon eine der wenigen verbrieften Quellen für den literarisch so oft behandelten Dr. Faustus, hätte dieser auf jene Ehrung wohl verzichtet, wenn er darauf Einfluss gehabt hätte (und wenn er denn tatsächlich gelebt haben sollte). Trithemius nämlich bezeichnet Faust als geschwätzigen Narr, eine verwegene Gestalt, die allenfalls Schläge verdiene. Paracelsus und Trithemius sind sich – wie auch der Altersunterschied annehmen lässt, Paracelsus wurde erst 1493 geboren – wohl nicht persönlich begegnet, allerdings

> "kann man von einer Reihe gleichgearteter geistiger Interessen der beiden vielseitig gebildeten, lese- und schreibfreudigen Männer ausgehen. Als Kenner der magia naturalis suchten sie die Offenbarungen Gottes in der Natur und in der pansophischen Ideengeschichte des geschriebenen und gedruckten Wortes." (Kuper 1998:113)

Eine sehr fruchtbare Begegnung ergab sich mit Heinrich Cornelius Agrippa von Nettesheim, der dem Würzburger Abt 1510 in seinem Kloster einen Besuch abstattete. Trithemius ermunterte seinen jungen Schüler, sein in Fragmenten vorliegendes Werk zur natürlichen Magie zu vollenden, die schließlich als „De occulta philosophia" im Druck erschien (Agrippa von Nettesheim 1533). Genauso wie den nachfolgenden Gelehrtengenerationen[19] waren Nettesheim die trithemischen kryptologischen Schriften wohlbekannt und Nährboden für eigene Entwicklungen. Die Methoden, die Trithemius in jenen weit rezipierten Schriften entwickelte, werden im nächsten Kapitel ausführlicher dargestellt.

5.3.2 Trithemius kryptographische Arbeiten

In der Vorankündigung zur Steganographia nimmt Trithemius die meisten seiner Verschlüsselungsmethoden, die er schließlich auf seine beiden kryptographischen Werke – die

19 Zu der man z.B. den Gegner der Hexenverfolgung Johann Weyer (1515/16-1588), den bereits erwähnten John Dee (1527-1608), die Kryptologen Giambattista della Porta (1535-1615) und Blaise de Vigenère (1523–1596), den Vater der Rosenkreuzer-Bewegung Johann Valentin Andreae (1586-1654) und schließlich auch den Empfänger des VM, Athanasius Kircher (1602–1680), zählen kann.

Steganographia und die *Polygraphia* – aufteilt, vorweg. Zum überwiegenden Teil handelt es sich dabei um Techniken, die scheinbar ohne Beispiel sind, gleichsam *creationes ex nihilo*. Das ist umso erstaunlicher, als dass die Verfahren eine bis dahin nicht gekannte Fülle und Vielfalt aufwiesen und auch – durch die Transpositionstafeln aus dem *Liber Quintus* der Polygraphia – eindrucksvolle Neuerungen brachten (vgl. 5.3.2). Strasser (1988b:55ff) weist allerdings darauf hin, dass sich Trithemius durchaus mittelalterlicher Anregungen bedienen konnte. So rezipierte er die Zeichenalphabete der Hildegard von Bingen genauso wie die *Tironischen Noten*[20]. Ferner hat seine polyalphabetische Transpositionstafel Ähnlichkeit mit Tabellen zur Berechnung von Feiertagen der Oster- und Fastenzeit, er konnte aus den kombinatorischen Arbeiten von Raymundus Lullus schöpfen und wurde ganz sicherlich durch die Buchstabenkombinatorik der Kabbala beeinflusst, mit der er sich als Schüler von Johann Reuchlin, dem ersten deutschen Hebraisten, der als Nichtjude die hebräische Sprache erlernte (*De verbo mirifico* 1494, *De arte cabbalistica* 1517) in Heidelberg auseinandergesetzt hat.

Der Begriff Kabbala geht auf die hebräische Wortwurzel *q-b-l* zurück, welche die Bedeutetungen *Überlieferung*, *Übernahme* und *Weiterleitung* trägt. Kern der kabbalistischen Tradition ist die Ansicht einer von Gott modellierten Entsprechung von Mensch und Universum, was auch Auswirkungen auf die Sprache hat: In der kabbalistischen Wortmagie etwa hat die Gestalt von Wörtern (Signifiant) Einfluss auf das Signifié.[21] Basierend auf dieser Theorie des sprachlichen Zeichens wurden innerhalb der kabbalistischen Tradition drei hermeneutische Verfahren entwickelt, die der Interpretation von Wörtern dienen sollten:

- Die **Gematrie** bildete die Buchstaben des hebräischen, später auch des griechischen Alphabets auf Zahlenwerte ab. Die ersten zehn Buchstaben des Alphabets, hebräisch *Aleph* bis *Jod* bekamen die Werte 1-10, der elfte (*Kaph*) den Wert 20, der daraufolgende (*Lamed*) den Wert 30 usw. Ganzen Wörtern wird dann die Summe der den Buchstaben entsprechenden Werten zugewiesen. Als kryptographisches Verfahren kann die Gematrie nicht angewendet werden, da die Berechnung nicht eindeutig rückführbar ist. Sie fand daher eher in der Mystik Verwendung.

20 Bei den Tironischen Noten handelt es sich um eine antike Kurzschrift, die auch zu Verschlüsselungszwecken eingesetzt wurde, vgl. Kahn 1996:71ff.

21 In der kurzen Darstellung zu Universalsprachen (Kapitel 4.3.2) wurde ausgeführt, dass auch die Entwickler solcher Sprachen oft genau diesen Einfluss zwischen diesen beiden (von der modernen Sprachwissenschaft, d.h. seit de Saussure 1916, als *arbiträr* zugeordnet angenommenen) Seiten des sprachlichen Zeichens in ihrem Sprachkonstrukt spiegeln wollten.

- Die drei Verfahren der **Temuhre** (*Atbash*, *Avgad* und *Albam*) sind Chiffren mit einfacher monoalphabetischer Substitution. Bei Atbash etwa wird der erste Buchstabe des hebräischen Alphabets (*Aleph*) durch den letzten (*Taw*) ersetzt, sowie der zweite (*Beth*) durch den vorletzten (*Sin*) usw.
- Das **Notarikon** schließlich nutzt Buchstaben-Transformationen, bspw. um ein Wort aus den Anfangs- oder Endbuchstaben der Wörter eines Satzes abzuleiten (Dieser spezielle Fall wird auch *Akronym* genannt). Das Verfahren ist ebenfalls nicht auf einen eindeutigen Klartext rückführbar und mithin nicht für kryptologische Zwecke einsetzbar.

Strasser (1988b:60ff) merkt an, welch großen Einfluss insbesondere die kabbalistischen Techniken auf die trithemischen Chiffrenentwürfe genommen haben. Allerdings wurden sie von Trithemius soweit modifiziert, dass sie auch für die geheime Informationsweitergabe und nicht ausschließlich zur Textexegese genutzt werden (wobei auch letzteres im Fall von Trithemius Schriften eine große Rolle spielt, s.u.). Die Modifikation kabbalistischer Praktiken für die Nutzung als kryptographische Methoden ist notwendig, da sich die beiden Bereiche zwar ähneln (schließlich operieren sie beide mit vergleichbaren Methoden auf Texten), die in der Kabbala beschriebenen Verfahren aber im Vergleich zu echten kryptologischen Verfahren extrem flexibel und spekulativ sind (Kahn 1996:92).

Die Chiffren der Steganographia

Wie in der Schilderung von Trithemius' Leben (5.3.1) dargelegt, begann die Rezeptionsgeschichte der Steganographia mit einer Art Missverständnis, das allerdings – zumindest teilweise – durch den unvorsichtigen und anmaßenden Text der trithemischen Vorankündigung ausgelöst wurde. Im ausgehenden Mittelalter wurde es offenbar als nicht besonders kritisch angesehen, von kosmosbestimmenden (selbstredend Gott untergeordneten) Planeten- bzw. Naturgeistern auszugehen. Trithemius aber verpackte seine Schlüssel, die auf den Klartext angewendet werden müssen, in Beschwörungsformeln, die Nichteingeweihten völlig unzugänglich waren. Damit lieferte Trithemius selbst die Munition, welche eingesetzt wurde, um ihn der Gotteslästerung zu bezichtigen.

Trithemius hatte ursprünglich angekündigt, die Steganographia würde aus fünf Büchern bestehen, er vollendet aber letztlich nur zwei; ein dritter Teil – von deutlich geringerem Umfang als die ersten beiden – erweckt den Eindruck eines Fragments. Er wurde oftmals als nicht zugehörig betrachtet, weil er stark vom Muster der ersten beiden Teile abwich

und es lange Zeit (bis 1996, also fast 500 Jahre; s.u.) unklar war, ob er überhaupt eine Verschlüsselungsmethode verbarg. Es existieren zwei Fassungen des Werkes,[22] einerseits eine allgemeinverständliche Frühfassung (*Clavis Steganographiae*, im Folgenden *Clavis*, entstanden 1498-1499), in der dem Leser die Auflösungen der Chiffren umgehend präsentiert werden, sowie eine inhaltlich erweiterte und verschlüsselte dreiteilige Zweitfassung (*Steganographiae libri tres*, entstanden 1500; im Folgenden abgekürzt mit S.I-S.III), die in ihrer Unverständlichkeit den oben beschriebenen Argwohn heraufbeschwor. In der späteren Fassung bestehen der erste Teil aus 32, der zweite aus 24 und der dritte aus nur einem Kapitel. Neben dem sehr unterschiedlichen Umfang weicht das dritte Buch auch inhaltlich und vom Aufbau her signifikant von den beiden ersten ab.

Steganographia I In der S.I werden verschiedene Verfahren beschrieben, eine geheime Botschaft zu tarnen, indem die signifikanten Buchstaben durch eine weit größere Menge von Blendbuchstaben (Nullen) maskiert werden.[23] Signifikante und Blendbuchstaben ergeben dabei zusammengenommen einen relativ unauffälligen lateinischen Text. Das Wissen darum, welche der Buchstaben nicht als Blende dienen, verschlüsselt Trithemius in einer arkansprachlichen[24] "Beschwörungsformel", die er unterschiedlichen Geisternamen zuordnet, bei denen er sich aus "am Rande des katholischen Glaubenszeltes tummelnden Engelschaaren [sic]" (Ernst 1996:70) bedient. Die Rekonstruktion des Textes erfordert dabei eine doppelte Dechiffrierung, die Trithemius in der Clavis, aber nicht in der späteren Version der Steganographia beschreibt: Die Beschwörungsformel enthält den zu dechiffrierenden Schlüssel, der dann auf den lateinischen Text angewendet werden muss, um die geheime Botschaft zu entschlüsseln.

Jedes Kapitel der ersten beiden Bücher der Steganographia ist mit einem der Geisternamen überschrieben. Die einleitenden Texte der Kapitel beschreiben auf lateinisch und auf sehr schwer durchschaubare Weise die Rangfolge einzelner Geister, die diesen Geis-

22 In der später gedruckten Fassung, die auch in Anhang E referenziert ist, finden sich beide Versionen.

23 Die Maskierung wird inzwischen unter den steganographischen Verfahren eingeordnet (vgl. Kapitel 5.1), deren Benennung mittelbar auf Trithemius zurückgeht: In der Zeit nach Trithemius wurde der Begriff Steganographie durch den Begriff Kryptologie verdrängt, bis ihn McCracken (1948) für verbergende Geheimschriften wieder vorschlug, was von Kahn (1996:xiii) übernommen wurde und seither der allgemeinen Begriffsdefinition entspricht (vgl. Ernst 2001:2).

24 Die Definition das Begriffes "Arkansprache" findet sich in Eis (1962:57): Sie "stellt einen wissenschaftlichen Sachverhalt oder eine praktische Arbeitsweise so dar, daß es nur dem Eingeweihten verständlich ist. Sie benützt dazu außer Geheimschriften und Geheimwörtern auch Symbole, Metaphern, fingierte Personen und mythologisierende Erzählungen von verwirrender Phantastik, wobei die Bedeutung der einzelnen Elemente von Fall zu Fall abgeändert werden kann."

tern zugeordneten Planeten sowie deren verschiedene Konstellationen und vieles mehr.[25] Trithemius entlehnt die Namen der Geister der Kaballa. Teilweise werden durch diese arkansprachlichen Erläuterungen tatsächlich Informationen über die Art der im Kapitel verwendeten Verschlüsselungstechnik transportiert, wobei diese sich einfacher aus der Beschwörungsformel, die sich an den einleitenden Text anschließt (s.u.), ablesen lassen. Letztere sind also nicht als reines Blendwerk konstruiert, dienen aber wohl der Einbettung der einzelnen Methoden in einen der kabbalistischen Tradition entlehnten Gesamtzusammenhang. Hinter einleitendem lateinischen Text und der kunstsprachlichen Beschwörungsformel findet sich in jedem Kapitel wiederum ein lateinischer Text (meist in Gestalt eines Briefes), der die geheime Information enthält. Abschließend wird in einigen Kapiteln noch eine weitere Beschwörungsformel angeführt, die ebenso wie die erste vom Geist der Kapitelüberschrift eingeleitet wird. Die Gliederung der einzelnen steganographischen Kapitel entspricht also dem folgenden Muster:

1. Kapitelüberschrift (Geistername als Namensgeber des Verfahrens)
2. Einleitender Text (lateinisch, inhaltlich arkansprachlich)
3. Beschwörungsformel (Geistername gefolgt von arkansprachlichen Phantasiewörtern, enthält Schlüssel zum Verfahren)
4. Unauffälliger lateinischer Text (meist in Gestalt eines Briefes, enthält die getarnte Botschaft)
5. Zweite Beschwörungsformel (inhaltlich der ersten Formel gleich, nicht in jedem Kapitel vorhanden)

Stellvertretend für die nicht gerade unkomplizierte Methode der doppelten Chiffrierung sei hier die Technik, die Trithemius in S.I, Kapitel VIII einführt, vorgestellt. Der der Methode zugeordnete Geist ist *Maseriel*, die erste der Beschwörungsformeln lautet:

> Maſeriel bulan lamodyn charnoty Carmephin iabrun careſathroin aſulroy beveſy Cadumin turiel buſan Seuear; almoſ lycadufel ernoty panier iethar care pheory bulan thorty paron Venio Fabelronthuſy.

Wie von Trithemius im Clavis beschrieben, können die Beschwörungsformeln stets mit dem gleichen Algorithmus aufgelöst werden: Nur jedes zweite Wort ist überhaupt signifikant. Erstes und letztes Wort sind in keinem Fall bedeutungstragend. Man extrahiere also jedes zweite Wort, angefangen mit dem zweiten und füge diese zu einer Buchstabenkette zusammen:

> bulancharnotyiabrunaſulroyCaduminbuſanalmoſernotyietharpheorythortyparon

25 Eine sehr ausführliche Darstellung des magischen Beiwerks der Steganographia findet sich in Shumaker (1982).

Nun extrahiere man aus dieser Kette jeden zweiten Buchstaben, wieder mit dem zweiten angefangen. Als Ausgabe erhält man:

uacantibuauryauibſnloentitapertotvno

Das erinnert zwar zunächst an Latein (vacantibus ohne s), wird aber im weiteren Verlauf völlig unverständlich. Eine solche Technik ist, wie sich hier zeigt, extrem abhängig von der korrekten Wiedergabe der Formel. Fügt man in der ursprünglichen Formel zwischen care und ſathroin sowie zwischen ly und cadufel jeweils ein Leerzeichen ein, und ersetzt man ni im vorletzten Wort durch m, so erhält man eine völlig andere Buchstabenkette:[26]

uacantibuſ tribvſ treS ualent ita per totVm –

Vacantibus tribus tres valent. Ita per totum.

Trithemius liefert die deutsche Übersetzung durch die zweite *Maseriel*-Beschwörungsformel, die sich im Anschluss an den lateinischen Brief befindet, gleich mit. Derselbe Algorithmus, angewendet auf

Maſeriel onear Camerſin, Cohodor meſſary lyrno balnaon greal, lamedon odiel, pedarnoy nador ianozauy chamyrin

ergibt

naCH dryn gelden dr -

Nach dreien gelten dr(ei)

Die in der Arkansprache versteckten Anweisungen beziehen sich immer auf die Anfangsbuchstaben des steganographischen Textes. Für den lateinischen Brief aus S.I/VIII bedeutet dies, dass immer drei signifikante Anfangsbuchstaben auf drei nicht-signifikante folgen. Der Text muss also in Dreierblöcke unterteilt werden, von denen man die ungeraden weglässt und die geraden zusammenzieht.

Omnipotenſ ſempiterne Deuſ bonorum remunerator aequiſſime, qui filium tuum, noſtri generiſ eſſe participem voluiſti, ut redimeret, diabolica inuidia noſ miſerrimos qui ſola benignitate redundanſ, forma noſtri ſuſcepit, incorruptam, ex flore virginaliſ uteri, archangelo ſancto Gabriele inſinuante, quod Virgo conceptura, beatiſſimo tuo Spiritui perpetua virgo permaneret, immaculata clarior hominib. angeliciſq ſpiritibuſ praementior. Genuit regem omnipotentem, Deum et hominem, ſantißima et reuerendiſſima Virgo Maria, viriliſ conſortii omnino neſcia, ſine dolore parienſ, ſine triſtitia vagientem Deum, hominemque, ſuſcipienſ, ſemper...

26 Bei genügend Operationen erhält man natürlich immer das gewünschte Ergebnis, hier lassen sich aber alle drei Änderungen durch die Beseitigung von Druckfehlern begründen, v.a. wenn man bedenkt, dass der Setzer des Buches der Arkansprache wohl nicht mächtig war.

Die Anfangsbuchstaben der Wörter, die nicht ausschließlich blendende Funktion haben, ergeben nach der Dreierblock-Formel zusammengestellt den folgenden Satz:

> branger dif brieff gibt ſich groſer conſt vſ –
>
> Überbringer des Briefs gibt sich großer Kunst aus.

Trithemius zeigt hier und auch in vielen anderen seiner Klartexte die Neigung, sich despektierlich über das zustellende Gewerbe zu äußern.[27] Ein Grund dafür könnte sein, dass er damit demonstrieren will, dass der Bote – trotz Kenntnis des Briefes – völlig ahnungslos ist, welche Informationen er da überbringt.

Steganographia II Auch die restlichen in S.I beschriebenen Verfahren sind auf dieselbe Weise lösbar: Nach der Interpretation der Beschwörungsformel ergibt sich der Schlüssel, der auf die Anfangsbuchstaben des lateinischen Briefes angewendet werden muss, um den Klartext zu erhalten.[28] Oberflächlich betrachtet setzt sich dieses Muster auch in S.II fort, allerdings sind die Klartexte, die auf diese Weise entstehen, nicht direkt interpretierbar, sie müssen zusätzlich noch einer monoalphabetischen Substitution unterzogen werden. Trithemius kombiniert hier also jeweils ein steganographisches mit einem kryptographischen Verfahren, um die Sicherheit (damit allerdings auch den Ver- und Entschlüsselungsaufwand) zu erhöhen. Aus der Reihe fallen die drei letzten Kapitel aus der S.II (XXII, XXIII und XXIV), die anstelle von Buchstaben ganze Wörter oder Silben im Tarntext verstecken, welche dann entweder vorwärts oder rückwärts zusammengesetzt werden müssen (vgl. Ernst 1996:29).

Steganographia III Wirkt die S.II wie eine logische Fortsetzung der S.I mit erweiterten Mitteln (zur Steganographie gesellt sich die Substitution), so fällt das dritte Buch völlig aus der Reihe. Lange Zeit galt es als unklar, ob es überhaupt von Trithemius verfasst wurde oder sich in der Zeit zwischen seiner Anfertigung und seinem Druck – zwischen

27 Vgl. dazu Ernst (1996:154), nebst Aufführung der ähnlichen Klartexte in Fußnote 656: "Dem konsternierten Leser stellt sich die prinzipielle Frage, aus welch unlauteren Gründen der Absender so gern nachweislich schlechte Menschen mit seinen Botengängen betraut."

28 In S.I-Kapitel IV, *Asesiel* etwa lautet die in der Beschwörungsformel versteckte Botschaft "post unam uacante due valent", nach einer Null folgen also zwei signifikante Buchstaben. Angewendet auf den lateinischen Text ergibt sich "ich vvil noch hint vmb [x]i an der Porten clopfen: wart min und lais mich in", was wohl eine Verabredung um 11 Uhr an der Hintertür sein soll. Das x wurde dabei für ein eigentlich vorkommendes CH (aus Christus) eingesetzt, in der Mitte des Klartextes wurde außerdem ein I (innerhalb von *umb*) weggelassen, das möglicherweise wieder auf einen Druckfehler zurückzuführen ist.

denen ja bekanntermaßen über 100 Jahre lagen, in denen es nur über handschriftliche Kopien weitergegeben wurde – durch welche Art und wen auch immer, hinzugekommen ist. Fern lag ein solcher Gedanke nicht – so besaß etwa Herzog August der Jüngere, der unter dem Pseudonym Gustav Selenus 1624 ein Handbuch zu dem damaligen Stand der Kryptologie veröffentlichte, ein Manuskript der Steganographia, in dem sich im letzten Kapitel der S.I noch 12 weitere Beschwörungsformeln finden. Herzog August hält diese Formeln selbst für eine Fälschung, womit er richtig lag, wie Ernst (2001) inzwischen nachgewiesen hat. Selbst wenn die S.III von Trithemius selbst stammen sollte, war doch unklar, ob sie überhaupt ein kryptographisches Thema behandelt oder bloß eine Art schwarzer Magie, Pseudowissenschaft oder hermetisches Gedankengut enthielt (vgl. Ernst 1996:130). Insgesamt weist die S.III einen sehr viel geringeren Umfang als die ersten zwei Bände auf (nur 21 der insgesamt 180 Seiten der dreiteiligen Steganographia gehören zur S.III). Die Methode, die Trithemius in ihr beschreibt, soll die Weitergabe von Informationen ohne Wörter, Bücher und Boten ermöglichen und auf den sieben Planeten sowie den 21 Geistern beruhen. Garniert ist das kurze Kapitel mit diversen Tabellen, die kosmologische Ereignisse in zeitlichen Abläufen zusammenstellen.

Im Jahr 1998, das Geheimnis der S.III war nun seit fast 500 Jahren scheinbar ungelöst, entschloss sich der Mathematiker Jim Reeds von der Firma AT & T, hinter den Zahlen dieser Tabellen nach einer Chiffre zu suchen, was ihm auch gelang (Reeds 1998). Er reichte die Arbeit bei der Zeitschrift *Cryptologia* ein, wo sie auch zur Veröffentlichung angenommen wurde. Der bedauernswerte Reeds selbst war es, der im Zuge intensiverer Recherche bemerkte, dass er nicht der erste war, dem die Dechiffrierung glückte. Vielmehr war schon der Germanist Thomas Ernst Anfang der 1990er Jahre auf die Lösung gestoßen und hatte diese bereits publiziert (Ernst 1996) – allerdings weitgehend unbemerkt von der Öffentlichkeit in der deutschsprachigen "Zeitschrift für Mittlere Deutsche Literatur und Kultur der Frühen Neuzeit", *Daphnis*. Ernst wies noch dazu nach, dass das Geheimnis schon 300 Jahre früher – 1674 durch Wolfgang Ernst Heidel – gelöst worden war. Zwar war bekannt, dass Heidel sich der Entschlüsselung rühmte, ihm wurde aber kaum Glauben geschenkt, da er seine Lösung ausschließlich in einer eigenen Chiffre kommunizierte, die allen Angriffsversuchen widerstand. Ernst, der als erster seit Heidel wieder in den Besitz des trithemischen Klartextes gekommen war, konnte auf dieser Grundlage seinen Angriff auf die Heidel'sche Chiffre erfolgreich gestalten.[29] Der trithemische Klartext aus der S.III

29 Auf die Frage, weshalb Heidel seine Lösung wiederum verschlüsselt hatte, antwortete Ernst "It was cryptological vanity" ("A Mystery Unraveled, Twice", New York Times, 14.4.1998). Dies lässt sich auch

weicht im übrigen qualitativ in keiner Weise von denen aus den ersten beiden Büchern ab, auch die obligatorische Botenbeschimpfung findet wieder ihren Platz:

> branger dis brieff ist ein boser Schalg vnd ein dieb hvet dich for eme er wirt dich anderst bedringen vnd schedigen –
>
> Überbringer des Briefes ist ein böser Schalk und ein Dieb. Hüte Dich vor ihm, er wird dich sonst bedrängen und schädigen.

Die Episode um die S.III zeigt, dass auch Chiffren, die weit vor den modernen, fast nicht mehr angreifbaren Verfahren entworfen wurden, erst in jüngerer Zeit gebrochen werden konnten. Ernst und Reeds fanden letztlich einen geeigneten Ansatzpunkt zum Angriff auf die Chiffre, indem sie die Tabellen durch eine Neuanordnung der Kolonnen in eine andere Form brachten. Dabei erkannten Sie, dass Trithemius lediglich eine Reihe gleichartiger monoalphabetischer Verfahren (Buchstaben werden dabei durch eine Reihe unterschiedlicher Zahlen ersetzt) anwandte. Durch die Nutzung sehr vieler unterschiedlicher Geheimtextalphabete betrat Trithemius den Pfad, der ihn schließlich zur polyalphabetischen Substitution führen sollte. Mit der Polygraphia kam er auf diesem Pfad ein ganzes Stück weiter voran.[30]

Die Chiffren, die Trithemius in seiner Steganographia vorstellt, sind zwar sehr innovativ, letztlich lässt sich aber nicht nachweisen, dass sie irgendwann auch praktisch angewendet wurden. Das liegt sicherlich einerseits daran, dass das Werk im 16. Jahrhundert nur in verschiedenen Abschriften vorlag, und, als sich dies durch den Druck 1606 änderte, es umgehend von der mächtigen katholischen Kirche verboten wurde. Andererseits sind die Chiffren aber auch nicht unbedingt einfach anzuwenden und hochanfällig für Schreibfehler oder Missinterpretationen, wie auch in obigem Beispiel gezeigt werden konnte. Es ist ein

ohne weiteres auf Ernst selbst beziehen: Er wählte statt einer kryptographischen Methode eine steganographische, indem er seine Lösung (erst Jahre nachdem er sie entdeckte) im Mediavistenmagazin *Daphnis* versteckte, statt sie im Zentralorgan der Kryptologen, der *Cryptologia* zu publizieren.

30 Es sei noch erwähnt, dass die Version, welche der Mathematiker Reeds ursprünglich veröffentlichen wollte, inklusive Appendix auf 28 Seiten Platz fand, während der Germanist Ernst für seine Darlegung derer über 200 benötigte. Das soll keine Wertung implizieren – beide Artikel sind phantastische Arbeiten, für die man gerne auch eine längere Lektüre in Kauf nimmt, v.a. wenn sie so gelehrsam ist wie die 200 Seiten von Ernst (vgl. auch Reeds 1999). Im Vergleich der beiden Darstellungen zeigt sich auch die unterschiedliche Herangehensweise von Natur- und Geisteswissenschaftlern, wobei die Analyse der numerischen Daten in den Tabellen auf die gleiche Art bewältigt wird. Ernst äußert sich nicht darüber, ob er einen Computer für die Analyse genutzt hat, Reeds erwähnt, dass die meiste Arbeit durch die Übertragung der Zahlenkolonnen (von Microfiche) in seinen Rechner verursacht wurde. Hier soll nur darauf hingewiesen werden, dass sie beide, Reeds und Ernst, ein Mathematiker und ein Germanist, das gleiche textprozessierende System hätten nutzen können, um ihre Analysen zu unterstützen.

hoher Aufwand damit verbunden, den Klartext zu tarnen, da die Zahl der Blendbuchstaben die der Klarzeichen um ein Vielfaches übersteigt. Es war wohl vor allem der Ruf der Steganographia als magisches Werk – durch die missverständliche Vorankündigung in die Welt gesetzt und durch die arkansprachlichen Beschwörungsformeln genährt – der zeitgenössische Gelehrte/Magier wie John Dee anzog, welcher sich eine Abschrift angefertigt haben soll (vgl. Ernst 1996:93f). Ähnlichkeit mit dem VM ist in den Chiffren nicht auszumachen (wenn man von den Phantasiewörtern der Beschwörungsformeln absieht). Verbleiben wir in der Hoffnung, diese im zweiten kryptographischen Werk des Trithemius, der *Polygraphia*, zu finden, welches im folgenden behandelt wird.

Die Chiffren der Polygraphia

Die *Polygraphia* (zu Deutsch etwa *Vielschrift*), besteht aus insgesamt sechs Büchern (*Polygraphiae libri sex*, nachfolgend abgekürzt als P.I – P.VI), wurde von Trithemius in den Jahren 1506-1508 verfasst und 1515/16 nochmals gründlich überarbeitet (vgl. Ernst 2001:513). Im Gegensatz zur Steganographia wurde sie viel früher verbreitet (gedruckt posthum 1518), obschon Trithemius' eigentliche Intention auch hier war, ihren Inhalt nur den Königen und Fürsten vorzubehalten, die als einzige verantwortungsvoll mit dem Wissen um sichere Verschlüsselung umgehen könnten (vgl. Strasser 1988b:51). Sie wurde von der katholischen Kirche auch nicht auf den Index gesetzt, was wohl vor allem daran gelegen haben mag, dass Trithemius in diesem Fall auf eine spektakuläre Ankündigung verzichtete, eine solche also auch nicht – wie der an Bostius adressierte Brief, s.o. – in die falschen Hände gelangen konnte. Ebenso ließ er das arkansprachliche Beiwerk weg, so dass zumindest in den ersten beiden Büchern der Polygraphia jeder Anschein des Geheimnisvollen vermieden wurde.

Das für das dritte Buch der Steganographia angekündigte Verfahren, mit dem man innerhalb von zwei Stunden das Lateinische lesen, schreiben und verstehen können soll, ohne es vorher auch nur im Mindesten beherrscht zu haben, wird in den ersten beiden Büchern (P.I und P.II) dargelegt. Allerdings entpuppt es sich eher als – durchaus durchdachtes – Chiffrierverfahren, denn als tatsächliches Lernprogramm. Dessen ungeachtet wird diese Methode später ins Französische (durch de Collange 1561) und Tschechische (durch Mnishowsky, um 1628) übertragen, um es tatsächlich als Lernverfahren für Französisch bzw. Tschechisch einzusetzen.[31] Auch als Chiffre wurde es oft kopiert und u.a. mit deut-

31 Vgl. Strasser (1988b:39) hinsichtlich de Collagne bzw. Davidsson (1959) hinsichtlich Mnishkovski.

schen Wörtern bzw. Phrasen nachgebildet, so druckt bspw. Herzog August (Selenus 1624) ein geändertes lateinisches Beispiel des Kryptologen Giambattista della Porta genauso ab wie ein anonymes Manuskript mit deutschen Wortlisten. Schließlich beauftragt Mitte des 17. Jahrhunderts der deutsche Kaiser Ferdinand III. den Jesuiten Atanasius Kircher (in dessen Archiv vermutlich auch das VM über 250 Jahre verschwand, vgl. 4.1.2) damit, eine Universalsprache zu entwerfen, welche die Kommunikation im habsburgischen Vielvölkerstaat vereinfachen sollte. Ferdinand verweist in diesem Zusammenhang auf die trithemische Polygraphia,[32] die Kircher dann auch zu seiner *Polygraphia Nova et Universalis* (Kircher 1663) inspiriert.

Polygraphia I & II Die Chiffriermethoden der ersten vier Bücher (P.I – P.IV) sind alle auf das gleiche Prinzip zurückzuführen: Die Ersetzung einzelner Buchstaben durch ganze Wörter. Das heißt nicht, dass im verschlüsselten Text jeder Buchstabe durch immer das gleiche Wort ersetzt wird – eine solche Methode wäre eine schlichte monoalphabetische Substitution und damit, wie oben dargestellt, selbst für ungeübte Kryptoanalytiker leicht zu brechen. Obendrein ergäbe sich ein dermaßen redundanter Text, der die Tatsache, dass in ihm eine verborgene Nachricht transportiert wird, wohl kaum kaschieren könnte. Stattdessen kann jeder Buchstabe durch eine ganze Reihe verschiedener Wörter ersetzt werden. Trithemius erstellte zu diesem Zweck Substitutionstabellen, deren Zeilen sämtliche Alternativen für die jeweils zu verschlüsselnden Buchstaben enthalten. Insgesamt enthält jede Tabelle 24 Zeilen für die Buchstaben des lateinischen Alphabets (Reihenfolge *a b c d e f g h i k l m n o p q r s t u x y z w*), Ersetzungen von Satzzeichen oder Wortzwischenräumen waren also nicht vorgesehen. In den beiden ersten Büchern der Polygraphia finden sich 383 (P.I) bzw. 308 (P.II) Listen (in der Tabellen-Terminologie oben: Spalten), bestehend aus jeweils 24 lateinischen Wörtern. In einer Spalte findet sich jeweils eine bestimmte Wortart, darüber hinaus sind diese „Reihen von Substantiven, Adjektiven, Verben, Adverbien; Partizipien oder Konjunktionen [. . .] so sorgfältig ausgewählt, daß jedes Wort aus einer bestimmten Liste mit jedem aus der vorhergehenden und folgenden einen zumeist annehmbaren Sinnzusammenhang ergibt.“ (Strasser 1988b:45)

Die ersten sechs Spalten aus P.I (hier abgebildet in Tabelle 5.3) mögen diesen Sinnzu-

32 Strasser (1988b:53) erwähnt, dass Ferdinand mit einer von seinem Lehrer durch eine von diesem selbst übertragene tschechische Abschrift der Polygraphia I unterrichtet wurde. Dieser Lehrer war wahrscheinlich niemand anders als der von Marci im Brief an Kircher erwähnte Dr. Rafael (Mnishkowski, s.o. und 4.1.3).

sammenhang illustrieren. Für eine Verschlüsselung des Wortes *Voynich*[33] könnte etwa die Wortfolge

Auctor gloriosus saluficans temporalia archangelis suis in supercelestibus amen.

verwendet werden, die wie ein Ausschnitt eines Gebets anmutet, weshalb das Verfahren auch später als **Ave-Maria-Chiffre** bezeichnet wurde.[34] Dem des Lateinischen mächtigen Leser mögen die mit diesem Schlüssel erzeugten Gebete holprig vorkommen, so dass er möglicherweise Argwohn schöpft. Blaise de Vigenère (1586:183) berichtet allerdings, dass zumindest die Türken mittels der Anwendung einer analogen Methode vom venezianischen Botschafter in Konstantinopel getäuscht worden sein sollen.

Polygraphia III & IV Während die Listen der ersten beiden polygraphischen Bücher lateinische Wörter als Chiffren für Buchstaben nutzen, finden sich in den Listen der Polygraphiae III (132 Spalten) und IV (117 Spalten) offenbar nur Phantasiewörter. Diese sind allerdings nicht willkürlich ersonnen und auf die Tabelle verteilt worden, sondern folgen einer bestimmten Systematik: In P.III sind alle Wörter einer Spalte auf eine Art Wortstamm zurückzuführen, der in den einzelnen Zeilen mit verschiedenen Endungen kombiniert wird. Diese Vorgehensweise wird in P.IV gespiegelt: Hier finden sich in den Spalten Wörter mit gleichen Endungen, die pro Zeile mit einem eindeutigen Stamm kombiniert werden. Erinnern die Wörter in P.III an lateinische Formen, so ähneln die aus P.IV teils hebräischen, teils griechischen Bildungen. Stellvertretend für beide Methoden finden sich in Tabelle 5.4 die an Flektionsparadigmen natürlicher Sprachen erinnernden Formen für die Stämme *pas-* und *mastr-* (aus P.III) sowie die Endungen *-on* und *-ech* (aus P.IV). Dem aufmerksamen Kryptoanalytiker mag bei genauerer Betrachtung der P.IV-Reihen auffallen, dass diesen ein recht einfaches steganographisches Prinzip zugrunde liegt: Die Wörter bestehen bis auf den zweiten Buchstaben nur aus Blendzeichen. Will man etwa das Wort "Voynich" auch mit dieser Methode verschlüsseln (und bediente sich dabei lediglich aus der *-on*-Reihe, obwohl noch 106 weitere zur Verfügung stehen würden), ergäbe sich

durion coridion tymon anydion siradon echaton christion.

33 V=U, Verschlüsselung nur der ersten sechs Buchstaben. Das *h* am Wortende hätte mit der hier nicht abgebildeten siebten Spalte der P.I verschlüsselt werden müssen, wo es dem Wort *Benignitas* entsprochen hätte.

34 Herzog August d. J. veröffentlichte eine Reihe von 14 Listen, die statt lateinischen deutsche Wörter enthielten. Mittels dieser Listen konnte man ein von Metrik und Rhythmus an das *Vater Unser* angelehntes Gebet als Geheimtext produzieren (vgl. Strasser 1988a:114f).

Spalte	P.I-1	P.I-2	P.I-3	P.I-4	P.I-5	P.I-6
a	Deus	clemens	creans	celos	ſanctis	celis
b	Creator	clementiſſimus	regens	celeſtia	electis	celeſtibus
c	Conditor	pius	conſervans	ſupercelestia	predilectis	ſupercelestibus
d	Optifex	piiſſimus	moderans	mundum	ſanctiſſimis	eternum
e	Dominuſ	magnus	gubernans	mundana	iuſtis	perpetuum
f	Dominator	excelſus	ordinans	homines	iuſtificatis	ſempiternum
g	Conſolator	maximus	ornans	humana	predeſtinatis	fecola feculorum
h	Arbiter	optimus	exornans	angelos	angelis	euum ſanctum
i	Judex	ſapientiſſimus	conſtituens	angelica	archanges	feculum
k	Illuminator	inuiſibilis	dirigens	terram	amatoribus	regno celoſum
l	Illuſtrator	immortalis	producens	terrana	cultoribus	altiſſimis
m	Rector	eternus	decorans	tempus	amicis	excelſis
n	Rex	ſempiternus	ſtabiliens	temporalia	apoſtolis	paradiſo
o	Imperator	glorioſus	illuſtrans	euum	prophetis	olympo
p	Gubernator	fortiſſimus	intuens	euiterna	diſcipulis	paradiſyacis
q	Factor	ſanctiſſimus	monens	omnia	martyribus	olympicis
r	Fabricator	incoprehenſibilis	confirmans	cuncta	ſanctificatis	fulgoribus
s	Conſervator	omnipotens	cuſtodiens	vniuerſa	dominationibus	felictitate
t	Redemptor	pacificus	cernens	orbem	dilectis	felicitatibus
u	Auctor	miſericors	diſcernens	aſtra	ciuibus	glorioſis
x	Princeptſ	miſericordiſſimus	illuminans	ſolem	ſeruis	honore
y	Paſtor	cunctipotens	fabricans	ſtellas	famulis	magnificentia
z	Moderator	magnificus	ſaluificans	vitam	miniſtris	luce perpetua
w	Saluator	excellentiſſimus	faciens	viventia	confeſſoribus	patriaceleſti

Tabelle 5.3: *Die ersten sechs Spalten der Ave-Maria-Chiffre aus der Polygraphia I. Vor der sechsten Spalte hat Trithemius jeweils ein* ſuis in*, nach der sechsten Spalte ein* amen *vorgesehen.*

Aus der aus Buch III stammenden Reihe lässt sich der Klartext nicht so problemlos aus dem verschlüsselten Text ableiten, weil dort ein so einfaches steganographisches Prinzip nicht anzutreffen ist. Dennoch scheinen bei der Erstellung der P.III-Substitutionstabellen vage Prinzipien eingehalten worden zu sein, die wir später genauer analysieren werden. Die Verschlüsselung des Wortes "Voynich" – abermals mit Rückgriff auf lediglich eine der 132 Stellvertreterspalten – ergibt mit der P.III

pasul pasos paser pasis pason pasi pasin.

Wie umgehend auffällt, ist es mit dieser Methode möglich, einen Klartext so zu verschlüsseln, dass die Chiffre den Eindruck erweckt, als bestehe sie aus einer Reihe von Flexionsformen des gleichen Wortes einer unbekannten Sprache. Damit nicht genug: Die

Spalte	PIII-5	PIII-15	PIV-8	PIV-15
a	pasa	mastra	baron	samelech
b	pase	mastre	abaron	ebramelech
c	pasi	mastri	ocaron	achalech
d	paso	mastro	adelon	adelmech
e	pasu	mastru	meron	nemelech
f	pasan	mastran	ofilon	afemelech
g	pasen	mastren	agion	ageselech
h	pasin	mastrin	chorion	thomelech
i	pason	mastron	libion	diralech
k	pasun	mastrun	akyron	akafelech
l	pasas	mastral	elychon	alanech
m	pases	mastrel	amaron	amalech
n	pasis	mastril	enorion	onamech
o	pasos	mastrol	morison	somelech
p	pasus	mastrul	aporion	apomelech
q	pasal	mastras	aquilon	aquifalech
r	pasel	mastres	armaon	tramelech
s	pasil	mastris	osarion	asomelech
t	pasol	mastros	atharon	stomelech
u	pasul	mastrus	cuburon	tumelech
x	pasar	mastraff	axion	axomelech
y	paser	mastreff	tymeon	pymelech
z	pasir	mastriff	azaron	ozyfelech
w	pasur	mastroff	puualon	iuuemelech

Tabelle 5.4: *Je zwei Spalten aus Polygraphia III und IV*

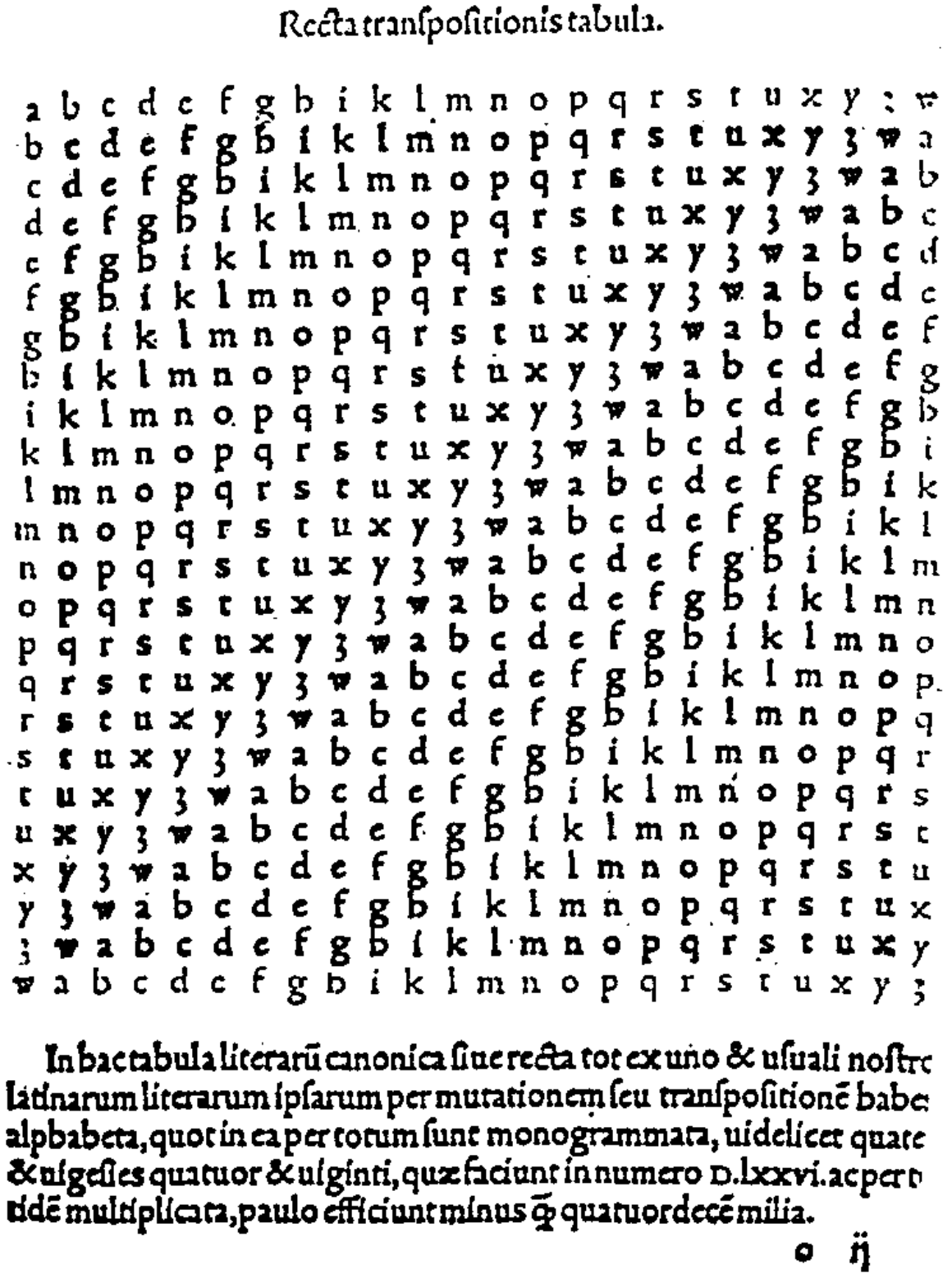
besondere Regelmäßigkeit im Aufbau der verschiedenen Substitutionschiffren gemahnt an die universalen Sprachmodelle, die von Wilkins, Dalgano und anderen erst über 150 Jahre später entworfen wurden (siehe 4.3.2). Diese bemerkenswerten Ähnlichkeiten zu zwei der Hauptauffälligkeiten des VM-Textes fordern eine eingehende Untersuchung dieser Verschlüsselungsmethode aus P.III, deren Aufbau in Anhang A einer ausführlichen Analyse unterzogen wird, zusätzlich werden im Folgenden mögliche kryptoanalytische Angriffsflächen auf diese Chiffre diskutiert. Vorher sollten aber der Vollständigkeit halber noch die Techniken der beiden letzten Bücher der Polygraphia vorgestellt werden.

Polygraphia V & VI In P.V findet sich direkt zu Anfang die berühmt gewordene Transpositionstafel[35] (siehe Abbildung 5.3). Zwar wurde die polyalphabetische Verschlüsselung genaugenommen zuerst durch Alberti (etwa 1466, also vier Jahrzehnte vor der Polygraphia) unter Nutzung einer Chiffrierscheibe ersonnen, mit Hilfe derer zwei Alphabete gegeneinander verschoben werden konnten. Allerdings hat die von Alberti beschriebene Methode den Nachteil, dass die Scheibe nicht nach jedem Buchstaben neu eingestellt werden musste, sondern erst nach der Chiffrierung von drei bis vier Wörteren – der Text lässt sich dadurch als Kaskade monoalphabetisch verschlüsselter Abschnitte analysieren und angreifen. Trithemius' System ist gegen diese Schwäche immun, weil es nach jedem Buchstaben das Geheimtextalphabet austauscht – so wird der erste Buchstabe mittels des ersten Alphabets substituiert, der zweite mit dem zweiten usw. Mit diesem Verfahren ist es möglich, wirklich polyalphabetisch verschlüsselte Geheimtexte zu erzeugen, die gegen jede Art von Häufigkeitsanalyse immun sind.

Zugegebenermaßen ist diese Methode, die auf einer starren Abfolge der Chiffrenalphabete basiert, zu statisch, um wirklich sichere Geheimtexte zu erzeugen. Sollten Unbefugte mit der Methode vertraut sein, so wäre für sie eine Dechiffrierung mit nicht viel mehr Aufwand verbunden, als eine monoalphabetische Chiffre zu brechen. Im weiteren Verlauf der P.V wird deshalb die Transpositionstabelle noch einmal rückwärts abgebildet, um zumindest verschiedene Abfolgen garantieren zu können. Zur Demonstration der Anwendung werden dann auch einzelne Zeilen der Transpositionstafel nochmals eigens aufgeführt. Trotz dieser Bemühungen gelingt es Trithemius aber nicht, sein durchdachtes polyalpabetisches System praktikabel zu gestalten. Erst de Vigenère 1586 sollte dies gelingen (siehe 5.2), weshalb mehrheitlich der französische Kryptologe als Erfinder der polyalphabetischen Methode angesehen wird. Neben den Passagen, die sich auf die Transpositionstafel beziehen, finden sich in der P.V außerdem noch tabellarische Aufstellungen für die Ersetzung von Buchstaben und Buchstabenbigrammen durch Zahlen. Dieses Verfahren scheint eindeutig durch die kabbalistische Gematrie (siehe oben, 5.3.2) beeinflusst worden zu sein.

Das sechste und letzte Buch der Polygraphia zeigt noch einmal eine qualitativ völlig neue Möglichkeit der Substitution von Klartexteinheiten auf: Die Ersetzung von Buchstaben durch Zeichen aus einem fremden (d.h. hier: vom lateinischen abweichenden) Alphabet. Trithemius führt einerseits das griechische Alphabet an, andererseits diverse fränkische, die er teilweise in Verbindung mit seiner gefälschten Hunibald-Chronik bringt.

35 Trithemius selbst nennt sie *tabula transpositionis*, in der heutigen kryptologischen Nomenklatur bezeichnet man eine solche Einrichtung eher als *Substitutionstafel*.



Recta tranſpoſitionis tabula.

a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w
b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a
c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b
d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c
e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d
f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e
g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f
h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g
i	k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h
k	l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i
l	m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k
m	n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l
n	o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m
o	p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n
p	q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o
q	r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p
r	s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q
s	t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r
t	u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s
u	x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t
x	y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u
y	z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x
z	w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y
w	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	x	y	z

In hac tabula literarũ canonica ſiue recta tot ex uno & uſuali noſtrę latinarum literarum ipſarum per mutationem ſeu tranſpoſitionẽ habe alphabeta, quot in ea per totum ſunt monogrammata, uidelicet quate & uigeſies quatuor & uiginti, quæ faciunt in numero D.lxxvi. acpe tidẽ multiplicata, paulo efficiunt minus q̃ quatuordecẽ milia.

o ij

Abbildung 5.3: *Die Trithemische* **Tabula Recta** *zur polyalpabetischen Substitution. Quelle: http://www.staff.uni-mainz.de/, zuletzt aufgerufen am 11.10.2011.*

5.3.3 Kryptoanalytische Angriffsflächen

In den letzten beiden Kapiteln wurden die teilweise sehr unterschiedlichen Chiffriersysteme aufgeführt, die der Abt Trithemius in seinen beiden kryptographischen Werken vorstellte. Dabei wurde angerissen, dass zumindest eines dieser Systeme – dasjenige, welches Trithemius in der P.III beschreibt – das Potential hat, aus einem beliebigen Klartext einen Geheimtext zu erzeugen, welcher dem Text des Voynich Manuskripts in bestimmten Auffälligkeiten ähnelt. Was genau diese Ähnlichkeiten ausmacht, wird in 5.4 formuliert und in 6.1 experimentell evaluiert. Vorher wird in diesem Kapitel dargelegt, welche Möglichkeiten sich für Kryptoanalytiker ergäben, sähen sie sich P.III-verschlüsselten Texten gegenüber, ohne dass sie auf die konkreten Ersetzungstabellen (das Codebuch) zurückgreifen könnten. Wäre eine Dechiffrierung solcher Texte möglich, oder von vornherein zum Scheitern verurteilt?

Das Verfahren der P.III steht bei Trithemius im Kontext der ersten vier Bücher der Polygraphia, die allesamt auf der Substitution von Buchstaben durch ganze Wörter beruhen. Infolgedessen werden hier auch alle vier Chiffriermodi auf ihre kryptoanalytischen Schwachpunkte überprüft. Dabei wird sich abzeichnen, dass diese oberflächlich betrachtet sehr ähnlichen Verfahren ausgesprochen unterschiedliche Aussichten hinsichtlich der Entschlüsselung ohne Codebuch aufweisen.

Allen vier Methoden ist gemeinsam, dass Trithemius für jeden zu ersetzenden Klartextbuchstaben eine Vielzahl von Homophonen erstellte und diese tabellarisch niederlegte. Die Zeilen dieser Tabellen repräsentieren die zu substituierenden Buchstaben, deren Homophone in den verschiedenen Spalten aufgetragen sind. Nun soll detaillierter betrachtet werden, ob den Tabellen eine Systematik zugrunde liegt, die eine Rekonstruktion allein durch Kenntnis der Substitutionswörter möglich machen würde. Im Vordergrund steht hier die Bestimmung der Zeilenzugehörigkeit, da man den verschlüsselten Buchstaben unmittelbar bestimmen kann, wenn es gelingt, die Zeile zu ermitteln, zu der ein Wort gehört. Die Kenntnis der Zeilenzugehörigkeit von Wörtern eines polygraphisch verschlüsselten Textes ist damit gleichbedeutend mit seiner Entschlüsselung. Die Bestimmung der Spaltenzugehörigkeit eines Wortes ist dagegen nachgeordnet. Aus ihrer Kenntnis ergibt sich vordergründig kein unmittelbarer Nutzen für die Entschlüsselung. Möglicherweise liefert sie aber zumindest Hinweise darauf, zu welcher Zeile ein polygraphisches Wort gehören könnte, womit sie wenigstens mittelbar für die Dechiffrierung nützlich wäre.

Die Frage, der hier nachgegangen wird ist also, ob es Anhaltspunkte dafür gibt, ohne

Rückgriff auf die tatsächlichen Substitutionstabellen, also rein aus der Kenntnis des trithemischen Systems, einen mit diesem System verschlüsselten Text dechiffrieren zu können. Trithemius selbst weist mehrfach darauf hin, dass Sender und Empfänger die gleiche Ausgabe der Polygraphia besitzen müssen, um über diese Methode kommunizieren zu können. Dies wurde auch als gravierender Nachteil des Systems angesehen und ist wahrscheinlich auch dafür verantwortlich, dass es keinerlei Anhaltspunkte für tatsächliche Anwendungen der trithemischen Systeme außerhalb der trithemischen Werke gibt.[36] Die Umständlichkeit der Systeme korreliert allerdings mit einer relativ hohen Sicherheit gegen eine unautorisierte Entschlüsselung. Ist die Methode über Substitutionswortlisten aber wirklich so sicher und damit wirklich so umständlich? In diesem Fall hätten ein Analytiker – fände er ein verschlüsseltes Dokument, das offenbar mit Wortsubstitutionstabellen chiffriert wurde, allerdings mit anderen Wortlisten als mit den in der Polygraphia abgedruckten, möglicherweise sogar mit nicht-lateinischen Buchstaben geschrieben – nicht die geringste Chance, dieses Dokument zu entschlüsseln. Doch betrachten wir die Rekonstruktionsmöglichkeiten der trithemischen Substitutionstabellen für die drei verschiedenen Verfahren – Phantasiewörter mit gleicher Endung (P.IV), Lateinische Wörter (P.I & P.II) und Phantasiewörter mit gleichem Wortstamm (P.III) im Einzelnen.

Angriff auf die Polygraphia IV

Die Rekonstruktion der Tabelle aus der P.IV ist elementar: Bei dieser Verschlüsselungsmethode bestimmt eindeutig der zweite Buchstabe die Zeilenzugehörigkeit, die weit weniger wichtige Zugehörigkeit zur Spalte ergibt sich nahezu[37] genauso eindeutig aus dem Suffix des Substitutionswortes. Damit genügt es völlig, die Verschlüsselungsmethode zu kennen (nur einer, und zwar immer der gleiche Buchstabe ist signifikant, alle anderen sind Blenden), um den Text zu entschlüsseln. Hilfreich wäre natürlich noch zu wissen, welcher Buchstabe genau der signifikante ist (Kenntnis des Schlüssels), es dürfte allerdings auch relativ schnell durch bloßes Ausprobieren klar werden, um welche Position es sich handelt.

Angriff auf die Polygraphiae I & II

Ganz anders verhält es sich mit den Tabellen, welche die Substitutionen für die Ave-Maria-Chiffre enthalten. Die Systematik ist hier relativ begrenzt: Um den codierten Text

36 Ausnahme ist die von de Vigenère erwähnte byzantinische Verwendung der Ave-Maria-Chiffre, s.o.
37 Trithemius lässt einige wenige Ausnahmen zu (vgl. Strasser 1988b:51).

in ein einigermaßen unauffälliges Gewand (in Gestalt eines lateinischen Gebetes) zu kleiden, war es für Trithemius lediglich[38] notwendig, die einzelnen Spalten so aufzufüllen, dass sich syntaktischer (kompatible Wortart, gleiche morphosyntaktische Eigenschaften) und semantischer Anschluss (kompatibles Bedeutungsfeld) an die Spalten rechts und links ergibt. Es ist also zu erwarten, dass sich innerhalb der Spalten weitgehend gleiche Wortarten mit gleichen morphosyntaktischen Eigenschaften befinden. Die Zugehörigkeit zu Spalten ließe sich also zumindest in bestimmtem Maße rekonstruieren. Leider gilt das aus Sicht der Kryptoanalyse sehr viel interessantere Zeilenzugehörigkeit nicht im Geringsten, zumindest konnten bisher keine Regelmäßigkeiten in der Abfolge innerhalb einer Spalte festgestellt werden. Selbst wenn man die Spaltenzugehörigkeit aufgrund morphosyntaktischer und semantischer Eigenschaften eindeutig bestimmen könnte, ergäbe sich als einziger Anhaltspunkt für die Kryptoanalyse, dass jedes Wort in der Spalte einen anderen Buchstaben substituiert als alle anderen Wörter der Spalte. Auch wenn Kryptoanalytiker für ihr Werk bisweilen sehr wenig Information benötigen: Eine Entschlüsselung auf dieser Basis wäre eine so gut wie unlösbare Aufgabe. Ohne den Schlüssel – hier die Wortlisten der Polygraphia – müsste man wohl vor der Ave-Maria-Chiffre kapitulieren.

Angriff auf die Polygraphia III

Betrachtet man die triviale Rückführung einer Wortform auf den substituierten Klarbuchstaben der Tabelle aus dem vierten Buch der Polygraphia auf der einen und die vermutlich unmögliche in den Tabellen der ersten beiden Bücher auf der anderen Seite, so scheint die Verschlüsselungsmethode aus der P.III eine Zwischenposition einzunehmen: Zwar lässt sich die Zeilenzugehörigkeit nicht aus einem einzelnen Buchstaben bestimmen, dafür aber scheint sich Trithemius beim Auffüllen der Tabelle mit den Wortchiffren an eine strenge Systematik gehalten zu haben. Diese ermöglicht es unter Umständen, Rückschlüsse von einem Wort auf den codierten Buchstaben zu erhalten, ohne auf die Wortlisten der Polygraphia zurückzugreifen: Die Wörter sind in den allermeisten Fällen zerlegbar in

1. einen Wortstamm, der vor dem letzten Vokal endet,
2. eben jenen letzten Vokal und
3. einen – fakultativen – abschließenden Konsonanten.

38 "Lediglich" ist hier ausschließlich auf die relativ simple Systematik bezogen. Es ist eine kaum angemessen zu würdigende Leistung von Trithemius gewesen, mehr als 6000 Wörter so anzuordnen, dass sich beim Verschlüsseln fast völlig unauffällige Gebete als Chiffre ergeben.

Durch den Wortstamm lässt sich die – wie oben angemerkt für sich allein genommen keine Information tragende – Spaltenzugehörigkeit des Wortes ermitteln. Die Kombination aus letztem Vokal und Schlusskonsonanten determiniert die Zeilenzugehörigkeit, durch die auf den Klarbuchstaben geschlossen werden kann. Leider gehorcht hier nur die Vokalverteilung einer strengen Symmetrie, so dass nicht genau *ein* möglicher Klarbuchstabe ausgemacht werden kann, sondern eine vier bis fünf Elemente große Menge. Beispielsweise lässt eine Form *Xan* darauf schließen, dass der durch sie verschlüsselte Klartextbuchstabe entweder ein *a*, ein *f*, ein *l*, ein *q* oder ein *x* ist. Wenn im verschlüsselten Text ebenfalls die Form *Xa* auftritt, so ist es wahrscheinlicher, dass diese das *a* codiert, also für *Xan* nur noch die Menge *{f,l,q,x}* übrig bleibt. Weiterhin bestehen Verbindungen zwischen Wörtern mit gleichem Stamm und Endkonsonanten: Hat man bspw. mit einiger Sicherheit ermittelt, dass sich hinter *Xan* das *f* verbirgt, so steht mit der gleichen Sicherheit fest, dass *Xen* den Buchstaben *g* codiert.

Die genauere Analyse der P.III (vgl. Anhang A) ergibt, dass sie einem oberflächlich regelmäßigem Aufbau unterliegt, sich im Detail betrachtet aber eine Reihe von Abweichungen in ihr finden. Ob Trithemius die Möglichkeit der Rekonstruktion seiner P.III-Tabelle intendierte, kann hier nicht abschließend geklärt werden. Einerseits geht er in der Vorrede zum dritten Buch der Polygraphia auch bei diesem Chiffriersystem davon aus, dass Sender und Empfänger identische Ausgaben der Substitutionstabellen vorliegen haben müssen, um über diese Methode zu kommunizieren. Andererseits erscheint es unplausibel, dass er die Regelhaftigkeit der Tabelle nur deshalb zugrundegelegt hat, um sie auf einfachem Wege mit unterschiedlichen Formen füllen zu können. Zumindest im Zeitalter der modernen Informationstechnologie sollte ein Entschlüsselungsversuch eines mit der P.III verschlüsselten Textes hinreichender Länge auch ohne Substitutionstabellen gelingen können.

5.4 Zusammenfassung und Überleitung

Ausgangspunkt für dieses Kapitel war die Suche nach einem Chiffrierverfahren, welches einen Geheimtext erzeugen kann, der dem VM-Text hinsichtlich der zuvor herausgearbeiteten Auffälligkeiten ähnelt. Zunächst wurde dafür eine allgemeine Einführung in die Grundbegriffe und die Geschichte der Kryptologie von ihren Anfängen bis zu den modernen Verfahren gegeben. Da das VM mit hoher Wahrscheinlichkeit irgendwann zwischen dem 15. und dem 17. Jahrhundert entstanden sein muss, wurde auf die in der frühen Neuzeit entwickelten Verfahren fokussiert. Als einer der produktivsten Entwickler innovativer

kryptographischer Methoden erwies sich zu dieser Zeit der Benediktinermönch Johannes Trithemius. Tatsächlich zeigt sich bei einer seiner Chiffren – bei derjenigen, welche er im dritten Buch der Polygraphia (P.III) beschreibt – dass sie einen VM-ähnlichen Klartext generieren kann. Die Annahme, dass zur Erzeugung des VM-Textes eine Methode ähnlich der in der P.III beschrieben verwendet wurde, in der einzelne Buchstaben durch verschiedene Phantasiewörter substituiert werden können, wobei diese Phantasiewörter einer partiell[39] geordneten Tabelle entstammen, wollen wir P.III-Hypothese nennen. In Tabelle 5.5 (Seite 151) werden nochmals sämtliche in 4.4 aufgelisteten Auffälligkeiten des VM-Textes aufgeführt und innerhalb der P.III-Hypothese interpretiert.

Wie aus der Aufstellung zu entnehmen ist, hat die P.III-Hypothese augenscheinlich eine Antwort auf alle gelisteten Auffälligkeiten. *Aus dem Bauch heraus* ließe sich daher behaupten, dass das System der P.III angewendet werden kann, um Substitutionstabellen mit VM-artigen Wörtern zu erstellen und damit eine Chiffre zu erzeugen, der tatsächlich ein Klartext zugrundeliegt. Diese Vorgehensweise *aufs Geratewohl* deckt sich genauso wenig mit den in Kapitel 2 gestellten Anforderungen an die textprozessierende Wissenschaft wie sie auf das textprozessierende Werkzeug aus Kapitel 3 angewiesen wäre. Da die vorliegende Arbeit aber genau auf diesen Elementen aufbaut, soll die P.III-Hypothese als eine Anwendung der Textprozessierung überprüft werden. Dies wird im nächsten Kapitel geschehen.

Die P.III-Hypothese impliziert nicht nur Fragestellungen, die in die Textprozessierung fallen. Da diese aber nicht Kern der vorliegenden Arbeit sind, können sie hier nur kurz angerissen werden. Sollte die Hypothese nicht falsifiziert werden, so ist der Kreis der möglichen Verfasser des VM zwar etwas eingeschränkter, die Frage nach seiner Herkunft aber noch nicht endgültig gelöst. Dass Trithemius seine Chiffre selbst modifizierte und das VM damit verfasst hat, ist wohl eher unwahrscheinlich, da seine bekannten Werke ansonsten völlig ohne Zeichnungen auskommen und im Allgemeinen nicht geheimgehalten, sondern publiziert wurden.[40] In Trithemius Biographie (5.3.1) wurde dargelegt, dass er,

39 Dass die Ersetzungstabelle der P.III nur partiell geordnet ist, macht den Unterschied zum Verfahren der P.IV aus, in deren Tabellen die Zeilenzugehörigkeit eines Wortes an einer bestimmten Buchstaben-Position im Wort festgemacht werden kann. Hier wird davon ausgegangen, dass dem VM kein so einfaches steganographisches Verfahren zugrunde liegt; erstens ist es unwahrscheinlich, dass es noch nicht entdeckt wurde, zweitens spricht die Gleichverteilung an Information in VM-Wörtern (vgl. 4.2.3) gegen ein solches Verfahren.

40 Gleichwohl hegte er bereits vor der Havarie mit der Steganographia eigentlich den Wunsch zur Geheimhaltung seiner Chiffren. Er war sich auch über die mögliche Rufschädigung, der er sich ausgesetzt sehen würde, bereits in der Vorrede zu seinem ersten kryptologischen Werk im Klaren: "Er wisse, daß es in der Zukunft viele geben werde, die das von ihm Geschriebene nicht verstehen könnten, Zuflucht bei

Nr.	**Besonderheit des VM**	**Erklärung auf Grundlage der P.III-Hypothese**
1	Die Zeichen entsprechen keinem bekannten Alphabet.	Trithemius experimentiert mit abweichenden Alphabeten in der P.VI
2-3	Die Zeichen sind zu Wörtern und Paragraphen gruppiert, keine Interpunktion	Die P.III besteht aus Wörtern, die für Buchstaben stehen. Paragraphen könnten für Wörter oder Sätze stehen; Interpunktion ist nicht vorgesehen
4-8	Die Häufigkeitsverteilung der Zeichen entspricht der NaSp – Die Verbundentropie ist extrem niedrig – Die Wortlängen sind eher kurz und binomialverteilt – Die Zipf-Verteilung der Wörter entspricht der von NaSp – Die Information ist innerhalb der Wörter annähernd gleichverteilt	Eine P.III-ähnliche Methode könnte vergleichbare Kennwerte aufweisen, zur Verifikation dieser Hypothese sind computationelle Analysen notwendig
9	Die Anzahl verschiedener Wörter ist sehr klein	Könnte durch die beschränkte Anzahl verwendeter Substitutionstabellen erklärt werden
10	Bestimmte Zeichen kommen nur in bestimmten Regionen innerhalb von Wörtern vor	Auch in der P.III werden nur bestimmte Buchstaben als Schlusskonsonanten genutzt, vorletzte oder letzte Position ist immer von Vokal besetzt
11-12	Die Wörter haben morphologisch einen sehr regelmäßigen Aufbau – Wörter unterscheiden sich oft nur durch einzelne Zeichen voneinander	Das sind auch zwei Haupteigenschaften der P.III-Wörter
13	Gleiche oder ähnliche Wörter treten in direkter Nachbarschaft zueinander auf	Ist in der P.III der Fall, wenn eine Spalte der Substitutionstabelle mehrfach hintereinander genutzt wird
14	Verschiedene Seiten scheinen in unterschiedlichen Dialekten verfasst	Könnte durch die Verwendung unterschiedlicher Substitutionstabellen erklärt werden
15-16	Zeilen bilden eine funktionale Einheit – Weitreichende Korrelationen deuten auf einen zufälligen Auswahlprozess hin	Könnte durch einen Auswahlprozess über die Substitutionstabellen erklärt werden

Tabelle 5.5: *Zusammenstellung der Auffälligkeiten des VM-Textes, Erklärung auf Grundlage der P.III-Hypothese. Die Abkürzung NaSp steht hier wieder für "natürliche Sprachen", die Nummerierung wurde aus der Aufstellung in 4.4 auf Seite 110f übernommen.*

wenn nicht gar unmittelbaren Kontakt, so doch großen Einfluss auf eine Reihe Gestalten des 16. Jahrhunderts hatte, die im Dunstkreis von Geheimwissenschaften, Okkultismus und Magie standen, dem auch der Erzeuger des VM entstammen könnte. Es kommt also jeder in Frage, der die Polygraphia eingehend studiert hatte und die Phantasie besaß, die Methodik leicht zu modifizieren. Pferdefuß der Hypothese ist natürlich, dass bisher keine Substitutionstabellen mit den VM-Wörtern aufgetaucht sind. Die Spekulationen hinsichtlich der Autorschaft sollen hier auch nicht weiter betrieben werden, stattdessen wird in den folgenden Analysen ausschließlich der Text selbst in das Zentrum der Betrachtung gerückt.

Beschimpfungen und Verleumdungen suchen und seine guten und gottgefälligen Studien verbotenen Künsten sowie abergläubischen Erfindungen zuschreiben würden." (Strasser 1988b:44).

Zwischen der Wissenschaft, wie sie die Öffenlichkeit erträumt oder die Medien feiern, und der Wissenschaft, wie sie die Forscher täglich praktizieren, besteht dieselbe Diskrepanz wie zwischen Heldensage und Reisetagebuch

(Ortoli, Witkowski: Die Badewanne des Archimedes)

Kapitel 6

Analysen: Methodik und Ergebnisse

In diesem Kapitel werden die einzelnen Bereiche, die in den vorausgehenden Teilen dieser Arbeit weitestgehend separat dargestellt wurden, miteinander verknüpft. Indem Tesla (Kapitel 3) als Plattform für Analysen zum Text des Voynich-Manuskripts (Kapitel 4) genutzt wird (gestützt auf Grundlagen der Kryptologie, Kapitel 5) wird demonstriert, dass sich das System für den Einsatz in einer auf die problemlose Nachvollziehbarkeit durchgeführter Analysen ausgerichteten computer- und communitybasierten Textwissenschaft (Kapitel 2) eignet.

Konkret geht es um die Demonstration der Einsatzfähigkeit des Frameworks Tesla für eine wissenschaftliche Gemeinschaft, die sich mit der Erforschung des VM-Textes auseinandersetzt. Dazu zählt, beliebige Analysen über textuelle Daten durchzuführen, indem bereits bestehende Werkzeuge genutzt, aber auch – sollte dies notwendig sein – neue integriert werden können. Die textprozessierenden Komponenten werden innerhalb von Workflows zusammengestellt. Die Workflows können dabei beliebig modifiziert werden, indem entweder konkurrierende Methoden erprobt oder einzelne Komponenten ausgetauscht bzw. abweichend konfiguriert werden können.

Durch die vorausgegangene Darstellung der Geschichte des VM-Textes, die wahrscheinlich bis mindestens ins 17. Jahrhundert zurückreicht und die seit Newbold & Kent (1928) dokumentiert ist, sollte klar geworden sein, dass sich bisher relativ wenige Anhaltspunkte für die Kryptoanalyse des Textes ergeben haben. So ist es noch immer umstritten, ob es sich überhaupt um einen Code, eine Chiffre oder einfach nur um eine sinnlose Aneinanderreihung unbekannter Zeichen handelt. Geht man von der Hypothese aus, der Text des VM wäre durch eine wie auch immer geartete Methode in einen verständlichen Klartext zu überführen, lassen sich die Analysen hinsichtlich des Anwendungsbereichs in drei Fragestellungen einteilen, die aufeinander aufbauen:

1. Suche nach dem Verschlüsselungsmechanismus – Welche Methode wäre in der Lage, einen ähnlichen Text wie den, der im Voynich Manuskript zu finden ist, zu generieren?
2. Suche nach dem Schlüssel – Für den Fall, dass ein solches Verfahren gefunden wird – wie ließen sich mögliche Schlüssel für dieses Verfahren rekonstruieren?
3. Suche nach dem Klartext – Wenn eine Option existiert, mögliche Schlüssel zu ermitteln, welchen Inhalt hat der Klartext, den sie generieren?

Wenn man als Prämisse voraussetzt, dass dem VM ein verschlüsselter Klartext zugrunde liegt, der durch ein einheitliches Verfahren generiert wurde, so können monopartite, monographische Verfahren so gut wie ausgeschlossen werden: Eine solche monoalphabetische Substitution wäre schon längst gebrochen worden. Eine polyalphabetische Verschlüsselung kann allerdings auch nicht angenommen werden, da die Häufigkeitsverteilung der Glyphen eindeutig gegen eine solche spricht. Übrig bleiben nur Verfahren, die von diesen beiden weitverbreiteten Methoden abweichen, vorzugsweise solche, die zur Zeit der möglichen Erstellung des VM bereits ersonnen waren. Einige von ihnen wurden im letzten Kapitel vorgestellt, eines erwies sich gar auf den ersten Blick als Verfahren, das möglicherweise eine Vielzahl der beobachtbaren und beschriebenen Auffälligkeiten des VM-Textes erklären könnte.

Genau dieses Verfahren – beschrieben im dritten Teil der Trithemischen Polygraphia (kurz: P.III) – wird im ersten Teil dieses Kapitels (6.1) – auf potentielle Indizien, die darauf hindeuten, ein vergleichbares Verfahren liege der Verschlüsselung des VM zugrunde, untersucht. Das Verfahren vom Typ P.III (im Übrigen genauso wie diejenigen aus den ersten beiden Teilen der Polygraphia) hat die unbequeme Eigenschaft, dass der Schlüssel nichts anderes ist als ein komplexes Codebuch, ohne das eine valide Ver- und Entschlüsselung nicht möglich erscheint. Legt man ein solches Verfahren dem VM zugrunde, so kann man sich entweder auf die Suche nach genau diesem Codebuch begeben oder man kann versuchen, ein solches Codebuch zu rekonstruieren. Mit Ansätzen zu einer solchen Schlüsselrekonstruktion befasst sich Kapitel 6.2. Mit der Identifikation eines möglichen Verfahrens sowie der Rekonstruktion möglicher angewendeter Schlüssel könnte dann ein erster Angriff auf den (weiterhin nur vermuteten!) Geheimtext des VM gestartet werden. Die Möglichkeiten dazu werden in Kapitel 6.3 kurz skizziert, wobei mögliche Anschlusspunkte für an diese Arbeit anknüpfende Forschungen im Mittelpunkt stehen.

Die drei Fragestellungen – Identifikation ähnlicher Chiffrierverfahren, Ansätze zur Rekonstruktion des Schlüssels, eine Angriffsmöglichkeit auf den Klartext – werden in den

nächsten Kapiteln ausführlich behandelt. Die Kapitel weisen jeweils den gleichen inneren Aufbau auf, jedes ist in folgende fünf Abschnitte gegliedert:

1. Beschreibung der Problemstellung und methodischer Ansätze zur Lösung
2. Formulierung der Hypothesen
3. Operationalisierung: Umsetzung der Methodik im Experimenten-Design
4. Vorstellung der für die Umsetzung benötigten Komponenten
5. Ergebnisse der Experimente

Der jeweils erste Abschnitt erläutert die oben schon kurz dargestellte Problemstellung für die experimentellen Analysen ausführlicher und stellt methodische Lösungsansätze vor. Diese Lösungsansätze basieren auf textanalytischen Methoden, die ursprünglich für anders geartete Anwendungsfälle entwickelt wurden. Im Zuge der Analysen werden sie auf das hier bearbeitete Anwendungsgebiet transferiert. Ein derartiger Transfer von Methoden ist einer der essentiellen Gedanken, die dieser Arbeit und der Entwicklung des Systems Tesla zugrunde liegen. Aus diesem Grund werden auch die Kontexte, für die jene Methoden entwickelt wurden, kurz angerissen. Im jeweils zweiten Abschnitt werden dann diejenigen Hypothesen formuliert, welche experimentell verifiziert werden sollen.[1] In den dritten Abschnitten werden die experimentellen Operationalisierungen der entsprechenden zu implementierenden Methodiken dargestellt. Da es sich um Experimente in Tesla handelt, werden in den vierten Abschnitten die dafür jeweils benötigten Komponenten vorgestellt. In den finalen Abschnitten werden schließlich die Ergebnisse der Experimente aufgeführt und diskutiert.

6.1 Die Suche nach dem Verschlüsselungsverfahren

Bei der Bestimmung des Verschlüsselungsverfahrens wird untersucht, ob sich die Ähnlichkeit zwischen dem Text des VM und einem mit der P.III-Methode verschlüsselten Text mit empirischen Daten fundieren lässt. Weisen beide Textarten ähnliche Eigenschaften auf, die sie von anderen Texten (z.B. natürlichsprachlichen oder zufallsgenerierten) unterscheiden, so gewinnt die Hypothese, dass es sich bei den Zeichen des VM um Chiffren handelt, gegenüber der, dass sie keine Inhaltsseite haben, an Gewicht. Letztere, vertreten

1 Kritisch rationalistisch (vgl. Popper 1935) kann man eine Hypothese strenggenommen natürlich niemals verifizieren. Man würde eher davon sprechen, dass die Falsifikation der Hypothese experimentell scheitern, oder andersherum, dass die entsprechenden Alternativhypothesen falsifiziert werden sollen.

durch Rugg und Schinner (vgl. 4.3.3) hatte ihren Ausgangspunkt darin, dass weder eine natürliche Sprache noch ein Chiffrierverfahren gefunden wurde, welche die statistisch zu erfassenden Abnormitäten des VM-Textes nachbildete. Wenn hier nachgewiesen werden kann, dass eine solche Methode doch existiert und zu einer Zeit vorhanden war, die nach wie vor für die Entstehung des VM infrage kommt (vgl. 4.1), spricht dies wieder mehr dafür, dass der VM-Text doch eine Inhaltsseite aufweisen könnte, was neue Angriffsbemühungen kryptoanalytischer Art rechtfertigen würde.

6.1.1 Problemstellung: Vergleichbarkeit von Chiffriermethoden

Will man die Ähnlichkeit bzw. die Divergenz unterschiedlicher Texte untersuchen, so benötigt man Verfahren, über die solche Ähnlichkeiten bzw. Divergenzen intersubjektiv ermittelt werden können. Im Bereich der Korpuslinguistik bzw. Korpusstatistik werden dafür bestimmte Kennwerte quantifizierbarer Eigenschaften eingesetzt, die im ersten der folgenden Abschnitte aufgeführt werden. Für den Bereich der Kryptoanalyse wurde die Berechnung von Koinzidenzwerten entwickelt, welche Rückschlüsse auf die Zugehörigkeit von Texten zu bestimmten Sprachen zulassen. Diese werden im zweiten Abschnitt vorgestellt. Abschließend wird die bereits in 4.3.3 erwähnte Random-Walk-Methode ausführlicher dargelegt, die – angewendet auf den VM-Text – Schinner (2007) Evidenz dafür lieferte, dass der Text keine Inhaltsseite aufweist.

Alle drei Methoden sollten an unterschiedlichen Texten erprobt werden – dazu gehören der VM-Text, natürlichsprachliche Texte unterschiedlicher Sprachen und schließlich ein mit der P.III-Methode verschlüsselter, natürlichsprachlicher Text.

Traditionelle korpusstatistische Kennwerte: Häufigkeit und Entropie

Die Textstatistik nutzt quantifizierbare Eigenschaften von Texten, um diese charakterisieren, vergleichen und klassifizieren zu können. Ein möglicher Anwendungsbereich ist z.B. die Klärung der Herkunft eines Textes (Epoche, Autor), etwa in der Literaturwissenschaft oder der forensischen Linguistik. Statistische Eigenschaften von Sprache werden mitunter aber auch genutzt, um Rückschlüsse auf das Sprachsystem, speziell auf seine Eigenschaften als texterzeugende Struktur, zu ziehen (vgl. Hřebíček & Altmann 1993). Zur Textstatistik zählen rein deskriptive Zählverfahren, etwa die Erstellung von Häufigkeitstabellen über Wortvorkommen oder Wortlängen, darauf aufbauend die Berechnung

der Kennwerte Mittelwert und Varianz.[2] Dazu begibt sie sich auf die Suche nach Mustern, z.B. Wiederholungen von Zeichen oder Zeichenketten, welche für natürlichsprachliche Texte charakteristisch sind – im Gegensatz zu zufallsgenerierten Texten. Die Häufigkeit charakteristischer Muster ist auch ein Ansatzpunkt für kryptoanalytische Angiffe auf – vor allem monoalpabetisch – verschlüsselte Texte.

Der Text des VM weist laut Literatur (vgl. 4.2.3) hinsichtlich der Zipf-Verteilung seiner Einheiten ähnliche, hinsichtlich der Wortlänge und bestimmter Entropie-Berechnungen abweichende Werte gegenüber natürlichsprachlichen Texten auf. Zusätzlich wurde festgestellt, dass die Anzahl unterschiedlicher Wörter relativ klein ist. Dies müsste sich in der Type-Token-Relation widerspiegeln, die das Verhältnis des Vokabularumfangs zur Textlänge angibt. Für die Berechnung dieses Wertes gibt es unterschiedliche Ansätze (vgl. Wimmer 2005:362f), er ist außerdem abhängig von der Textlänge.[3] Bei der Durchführung der Experimente ist deshalb darauf zu achten, dass die Vergleichswerte entweder von Texten gleicher Länge stammen oder dass längennormalisierte Werte[4] herangezogen werden.

Koinzidenzwerte

Der Begriff der *Zeichenkoinzidenz* wurde von Friedman (1922) eingeführt. In seiner allgemeinen Form vereinigt er vier Funktionen unter sich, die mit den griechischen Buchstaben κ (Kappa), χ (Chi), ψ (Psi) und ϕ (Phi) bezeichnet werden. Die Funktionen, allen voran κ, fanden weite Verbreitung unter Kryptoanalytikern und führten u.a. dazu, dass polnische Kryptologen das Verschlüsselungssystem der deutschen Enigma angreifen konnten.[5] Über Koinzidenzwerte lässt sich auch auf eine sehr einfache Weise ermitteln, ob Texte hinreichender Länge zur selben Sprache gehören, da die Werte innerhalb von Sprachen relativ invariant sind, während sie sich für verschiedene Sprachen mehr oder weniger stark unterscheiden (s.u.).

Der Wert von κ gibt die relative Häufigkeit von zeichenweisen Übereinstimmungen zwei-

2 Sie verfolgt darüber hinaus aber auch "analytische Ansprüche und sucht eine 'verborgene Ordnung' [...] in Texten."(Schmitz 2000:196)

3 Die Wahrscheinlichkeit, dass das nächste Token zum ersten Mal auftritt, ist bei kürzeren Texten höher als bei längeren Texten.

4 Hier bietet sich z.B. der MTLD (measure of textual lexical diversity) von McCarthy (2005), auch als Standardisierte Type-Token-Relation (STTR) bezeichnet, an.

5 Über die Koinzidenzanalyse konnte erschlossen werden, dass die ersten sechs Buchstaben jeder mit der Enigma verschlüsselten Nachricht offenbar eine Art von Spruchschlüssel waren, der den eigentlichen Schlüssel – die Walzenstellung der Maschine – festlegte. Näheres findet sich bei Pommerening (2008).

er übereinandergelegter Texte an (vgl. Tabelle 6.1). Für Texte, in denen die einzelnen Elemente (Buchstaben bzw. Zeichen) gleichverteilt sind, ergibt sich für Kappa der Wert $1/n$ (n ist die Anzahl der unterschiedlichen in der jeweiligen Sprache verwendeten Zeichen). Da die Buchstabenhäufigkeit natürlicher Sprachen keiner Gleich-, sondern einer Zipfverteilung entspricht, bewegen sich die Kappa-Werte zwischen $1/n$ und 1.[6]

Text 1	e	i	n	b	e	i	s	p	i	e	l	t	e	x	t	l	i	e	d
Text 2	w	e	i	t	e	r	e	r	b	e	i	s	p	i	e	l	t	e	x
Index					*					*						*		*	

Tabelle 6.1: *Beispiel für die Berechnung der Zeichenkoinzidenz zweier Texte: Alignieren der Texte, Zählen der Übereinstimmungen (in diesem Beispiel 4), relative Häufigkeit der Übereinstimmungen ermitteln (hier:* $4/19 \approx 21\%$*, Kappa läge damit bei 0,21).*

Die Abweichung in der Häufigkeitsverteilung bestimmter Zeichen wird durch das Koinzidenzmaß χ angegeben. Das χ zweier Texte (T, T') definiert sich durch die Summe des Produkts der einzelnen Buchstaben-Vorkommen aus den Texten (m und m') geteilt durch das Quadrat der Anzahl aller Buchstaben (M); N ist hierbei die Anzahl der unterschiedlichen Buchstaben:

$$\chi(T, T') = \frac{\sum_{i=1}^{N} m_i * m'_i}{M^2} \tag{6.1}$$

Während der Kappa-Wert eines Textes sowohl im Falle einer monoalphabetischen als auch einer polyalphabetischen Chiffrierung invariant bleibt, verändert sich der Chi-Wert nur bei der monoalphabetischen Chiffrierung nicht. Dies ist auch der Grund, weshalb die Häufigkeitsanalyse polyalphabetische Chiffren nicht anzugreifen vermag. Die Berechnung von Kappa kann aber durchaus Rückschlüsse auf die Art polyalphabetischer Chriffrierungen zulassen. So kann etwa die Länge der Periode bestimmt werden (vgl. Bauer 2000:315ff).

Ein Spezialfall von χ ist, wenn $m = m'$, d.h. wenn der Text der gleiche ist. Dieser Wert wird auch als ψ bezeichnet. Er ist ein Maß für die Verteilung der Buchstabenhäufigkeit innerhalb einer Sprache. Statt des ψ wird in der Literatur häufig ϕ verwendet. Die Berechnung der beiden Werte ist nahezu identisch, bei der Berechung von ϕ wird lediglich dem Umstand Rechnung getragen, dass bei zwei identischen Texten die gleiche Position

6 Der Maximalwert 1 wird nur erreicht, wenn der Text eine Aneinanderreihung des immer gleichen Zeichens ist.

nur einmal gezählt wird (was zur Folge hat, dass die Summe der Buchstabenhäufigkeit um eins heruntergezählt wird):

$$\psi(T) = \chi(T,T) = \frac{\sum_{i=1}^{N} m_i * m_i}{M^2} \tag{6.2}$$

$$\phi(T) = \frac{\sum_{i=1}^{N} m_i * (m_i - 1)}{M * (M - 1)} \tag{6.3}$$

Jede Sprache hat charakteristische Werte für die vorgestellten Koinzidenzmaße, sie spiegeln auch zum Teil das Maß der Redundanz der jeweiligen Sprache wieder. Das Deutsche etwa hat mit ($\kappa \geq 0,075$) einen höheren Kappa-Wert als das Englische ($\kappa \leq 0,068$); das VM sollte aufgrund seiner Redundanzen deswegen einen eher hohen Kappa-Wert haben, ebenso ein P.III-Text. Hinsichtlich $\chi/\psi/\phi$ sollten sich diese letztgenannten Texte nicht wesentlich von natürlichsprachlichen unterscheiden.

Weitreichende Korrelationen

Zur Charakterisierung von Zeichenketten können u.a. Untersuchungen zu Korrelationen zwischen einzelnen Elementen der Kette herangezogen werden. Lokal begrenzte Korrelationen können z.B. durch Markov-Ketten[7] untersucht werden, für Studien weitreichender Korrelationen (englisch *Long Range Correlations*, kurz LRC) eignet sich u.a. der Einsatz eines eindimensionalen *Random-Walk-Modells*. Dieses wurde bereits für die verschiedensten Textsorten eingesetzt, zunächst für genomische Daten (Peng *et al.* 1992), später dann auch für durch Menschen generierte Texte (genauer für literarische und Programmiersprachen-Texte, siehe Schenkel *et al.* 1992 sowie Kokol *et al.* 1999). Die Arbeit von Schinner (2007) thematisiert die Eigenschaften des VM hinsichtlich LRCs.

Der eindimensionale Random Walk ist ein Bernoulli-Prozess (eine Folge von unabhängigen Bernoulli-Versuchen), das Resultat eines Random Walks über zufällige Daten ist eine Binomialverteilung. Um eine eindimensionale Zufallsbewegung auf einem Text durchzuführen, muss dieser zunächst in eine binäre Repräsentation überführt werden. Dafür wird jedes Element des dem Text zugrundeliegenden Alphabets durch eine eindeutige Bitfolge ersetzt. Im Falle genomischer Daten genügt eine zwei-Bit-Folge (womit vier verschiedene

7 Eine Markov-Kette ist ein stochastischer Prozess, der sich dadurch auszeichnet, dass die Vorhersage zukünftiger Zustände sich auf eine kleine Zahl (im Extremfall einen) vorhergehender Zustände beschränken lässt. Eine bekannte Anwendung ist der PageRank-Algorithmus von Google (U.S. Patent 6.285.999), der als Markov-Prozess definiert ist.

Zustände für die vier Nucleotidbasen Adenin, Thymin, Guanin und Cytosin dargestellt werden können)[8], im Falle eines Textes auf Basis des lateinischen Alphabets werden mindestens fünf Bits benötigt, womit 32 Bit-Kombinationen für 26 Buchstaben zur Verfügung stehen – oft werden die letzten sechs freien Bitsets mit sprachspezifischen Umlauten oder auch Interpunktionszeichen belegt.

Liegt der Text als Bitfolge vor, so kann er wie folgt durchlaufen werden: Der Random Walk startet bei 0, je Bit wird ein Schritt in die vorher festgelegte Richtung gemacht, gemeinhin +1, falls das Bit 1 und -1, falls das Bit 0 ist (vgl. Tabelle 6.2).

Wort	w					a					l				
Bits	1	0	1	1	0	0	0	0	0	0	0	1	0	1	1
Walk	1	0	1	2	1	0	-1	-2	-3	-4	-5	-4	-5	-3	-4
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tabelle 6.2: *Beispiel für einen Random Walk: Jeder Buchstabe wird durch eine Folge von fünf Bits ersetzt (a durch 00000, b durch 00001 usw.). Im Walk wird für jedes Bit 1 ein Schritt vor (+1), für jedes Bit 0 ein Schritt zurück (-1) gesetzt.*

Interessant sind dabei nicht die Werte des Walks selbst, sondern die gemittelten Abweichungen, die sich zwischen den verschiedenen Intervallen (l) des Walks ergeben. Um ein Beispiel zu geben: l_5 etwa bezeichnet das Intervall der Länge 5; in obigem Beispiel wäre das genau der Abstand zwischen den Bits zweier aufeinanderfolgender Buchstaben. Der Funktionswert $y(l)$ ist die Summe der Schritte $u(i)$ innerhalb dieses Intervalls:

$$y(l) \cong \sum_{i=1}^{l} u(i) \tag{6.4}$$

Je nach Ausgangsposition l_0 des Intervalls l ergeben sich unterschiedliche Werte für die Quantität $\Delta y(l)$:

$$\Delta y(l) \equiv y(l_0 + l) - y(l_0) \tag{6.5}$$

Im Beispiel aus Tabelle 6.2 etwa ergibt sich als Abweichung des Intervalls $\Delta y(5)$ mit $l_0 = 1$ der Betrag aus der Differenz der Werte des Walks aus den Spalten 1 und 6:

8 Tatsächlich fasst die erste Anwendung des Random-Walk-Modells auf genomische Daten (Peng *et al.* 1992) Adenin und Guanin als *Purine*, Thymin und Cytosin zu *Pyrimidinen* zusammen, es wird also nur ein Bit zur Unterscheidung benötigt.

$|1-0| = 1$. Die aussagekräftige statistische Quantität eines Walks ist dann die Varianz über die durchschnittlichen Abweichungen für die Intervalle (der Durchschnitt wird über alle l_0 berechnet, für das obige Beispiel wären das die Abweichungen $\Delta y(5) : |1-0| = 1$, $|0-(-1)| = 1$, $|1-(-2)| = 3$ usw.). Die Gesamtvarianz $F^2(l)$ ist dann definiert als die Differenz zwischen dem Durchschnitt der quadrierten Abweichungen und dem Quadrat der durchschnittlichen Abweichungen:

$$F^2(l) \equiv \overline{[\Delta y(l)]^2} - \overline{[\Delta y(l)]}^2 \tag{6.6}$$

Die Wurzel aus der Gesamtvarianz $F^2(l)$, also die Standardabweichung $F(l)$, wird tendenziell mit steigendem l größer. Interessant ist dabei, in welchem Maß der Funktionswert zunimmt. Finden sich in den Daten keine Korrelationen mit bestimmter Länge, so besteht nach Peng *et al.* (1992) zwischen l und $F^2(l)$ das folgende Verhältnis:

$$F(l) = l^\alpha \Leftrightarrow \alpha = \frac{log(F(l))}{log(l)} \tag{6.7}$$

Aus der Umformung im rechten Teil der Formel ergibt sich, dass α die Steigung der Funktion auf einer log-log-Skala ausdrückt. Bei einer Zufallsfolge würde sich für α ein Wert nahe 0,5 ergeben. Finden sich weitreichende Korrelationen, so ergibt sich ein Wert für α, der signifikant von 0,5 abweicht.

Peng *et al.* (1992) vergleichen DNA-Stränge mit und ohne Introns,[9] wobei sie beobachten, dass DNA ohne Introns keine weitreichenden Korrelationen aufweist ($\alpha \sim 0,50$), DNA mit Introns aber sehr wohl ($\alpha \sim 0,61$).

Studien weitreichender Korrelationen in von Menschen generierten Texten finden sich z.B. bei Schenkel *et al.* (1992) sowie Kokol *et al.* (1999). Beide untersuchen sowohl natürlichsprachlich verfasste Texte als auch Computerprogramme verschiedener Programmiersprachen. Beide Studien zeigen, dass die größere Formalität von Computerprogrammen nachweisbare LRCs zur Folge hat – bei Schenkel *et al.* (1992:54) weisen bereits compilierte Programme (also Maschinencode) eine Steigung von $\alpha \geq 0,9$ auf, Kokol *et al.* (1999) zeigen, dass auch uncompilierter Code deutlich von $\alpha = 0,5$ abweicht, allerdings um bestenfalls 0,15 Punkte. Die Ergebnisse für natürlichsprachliche Texte sind bei Schenkel *et al.* ebenfalls durchweg höher ($0,56 \leq \alpha \leq 0,85$) als die bei Kokol *et al.*($0,47 \leq \alpha \leq 0,58$).

9 Introns sind nicht-codierende Abschnitte innerhalb von Genen, die vor dem Translationsprozess entfernt werden. Peng et al. untersuchen einerseits Viren mit Introns sowie Intron-enthaltende Gene, andererseits Intron-lose Viren und synthetisierte cDNA (ohne Introns) auf LRCs.

Wenn überhaupt, dann lassen sich bei Kokol *et al.* nur sehr schwache LRC in natürlichsprachlichen Daten nachweisen. Dies ist auch das Ergebnis, was Schinner (2007) für natürlichsprachliche Texte erhält, woraus er schließt, dass sich der Prozess zu jedem Zeitpunkt des Random Walks in etwa gleich schlecht vorhersagen lässt. Für den VM-Text stellt er fest, dass kurze Passagen wie eine zufällige Reihe von Bits wirken ($\alpha \sim 0,5$), also natürlichsprachlichen Texten in dieser Hinsicht ähneln. Interessant erscheint dagegen die Entdeckung Schinners, dass nach etwa 360 Bits (was 72 Zeichen und damit etwa einer VM-Textzeile entspricht), der Zufallsprozess in einen vorhersagbareren Prozess mit ($\alpha \sim 0,85$) übergeht.[10] Während der Text also innerhalb einer Zeile zufallsverteilt sei, beständen über mehrere Zeilen hinweg Korrelationen.

Schon die sehr verschiedenen ermittelten Werte für die Steigung α bei natürlichsprachlichen Texten verlangen nach nachvollziehbaren Experimenten über solche Daten. Weiterhin sollte versucht werden, die Schinnerschen Werte für das VM innerhalb eines Experiments zu reproduzieren und mit denen eines P.III-Chiffrentexts zu vergleichen.

6.1.2 Hypothesen

Die zentrale Hypothese hinsichtlich der Suche nach dem Verschlüsselungsverfahren ist die, dass der Text des Voynich-Manuskripts (VM) über ein Chiffrierverfahren generiert wurde, das dem von Trithemius in der Polygraphia liber tertii (P.III) beschriebenen Methode ähnlich ist. Das heißt, dass die VM-Wörter für einzelne Klartextbuchstaben stehen und dass für jeden solchen Klartextbuchstaben eine große Zahl von Homophonen existiert. Zur Überprüfung der Hypothese muss gezeigt werden, dass sich über die P.III-Methode ein Klartext verschlüsseln lässt, der dem VM-Text ähnlich ist, vor allem bei den Merkmalen, die den VM-Text von natürlichsprachlichen Texten (NaSp) unterscheidet. Operationalisiert wird die Feststellung dieser Ähnlichkeit über die im Folgenden aufgezählten Merkmale:

1. Type-Token-Relation: Sollte im VM- und P.III-Text signifikant niedriger sein als bei NaSp-Texten (d.h. es sollten verhältnismäßig wenige Types im Verhältnis zu Tokens vorkommen).
2. Wortlängen: Bei VM- wie bei P.III-Text binomialverteilt, für NaSp-Texte trifft dies nicht zu (Es besteht Uneinigkeit darüber, durch welche Verteilung sich NaSp-

10 Dies gilt für Texte mit dem Currier A-Dialekt, Currier B weist auch hier eine Abweichung auf, erst nach ca. 3 Textzeilen wird der Text vorhersagbarer.

Wortlängen am besten beschreiben lassen, eine Binomialverteilung ist es aber definitiv nicht, vgl. dazu Grzybek 2006).

3. Wortentropie: Bisher beobachtet wurde, dass die Wortentropie im VM ungefähr der von NaSp entspricht. Ein P.III-Text sollte davon nicht signifikant abweichen.
4. Verbundentropie: Bei VM und P.III signifikant niedriger als bei NaSp.
5. Koinzidenzwerte: Kappa bei VM und P.III signifikant niedriger als bei NaSp.
6. Anzahl der Minimalpaare (Wörter, die sich nur in einem Zeichen unterscheiden): Bei VM und P.III signifikant häufiger als bei NaSp.
7. Wortzwillinge und -drillinge: Bei VM und P.III signifikant häufiger als bei NaSp.
8. Weitreichende Korrelationen (LRC): Die Steigung der auf einer log-log-Skala aufgetragenen Intervall-Standardabweichungen kann bei einer bestimmten Anwendung des P.III-Verfahrens dem des VM-Textes entsprechen.

6.1.3 Design der Experimente

Die Untersuchung der oben aufgestellten Hypothesen verlangt die Ermittlung der aufgeführten Werte für die unterschiedlichen Textdaten (VM, P.III und NaSp). Während der Text des VM und natürlichsprachliche Texte maschinenlesbar vorliegen, gilt dies für einen P.III-verschlüsselten Text nicht. Ein solcher muss also erst generiert werden, dabei sollte diese Generierung auf nachvollziehbare Weise erfolgen, d.h. durch die Verschlüsselung eines natürlichsprachlichen Textes über ein Werkzeug, welches das P.III-Verfahren implementiert und welches innerhalb der dokumentierenden Laborumgebung von Tesla prozessiert. Konkret werden die Eigenschaften der folgenden Texte experimentell erfasst:[11]

- Der Text des Voynich-Manuskripts in Form des Voynich Interlinear Archive File (Voynich IAF, vgl. Stolfi 1998 und 4.2.2). Aus diesem können verschiedene Transkriptionen und Sektionen extrahiert werden. Für die unten aufgeführten Experimente wurden die einzige vollständige Transkription – die von Takahashi – gewählt und alle Sektionen untersucht. Als Codierung wurde das EVA-Alphabet gewählt,

11 Bei der Auswahl der NaSp-Vergleichstexte spielten mehrere Kriterien eine Rolle. Die Texte sollten 1. aus der Zeit stammen, in der das VM vermutlich entstand (spätes Mittelalter bzw. frühe Neuzeit), sie sollten 2. in einer in Mitteleuropa zu der Zeit gesprochenen Sprache verfasst sein und 3. öffentlich (hier: über die Webseite des Projekts Gutenberg) verfügbar sein. Ausnahmen bilden der Text von Paracelsus (der nicht mehr öffentlich zugänglich ist, der aber aufgrund der Beziehung von Trithemius und Paracelsus dennoch aufgenommen wurde) sowie Caesars "De Bello Gallico" (um evtl. Parallelen zu altertümlichem Latein untersuchen zu können).

vereinzelt wird im Text auf abweichende Werte bei Nutzung des Currier-Alphabets eingegangen.[12]

- Als frühneuzeitlicher englischer Vergleichstext: John Dee "The Mathematicall Praeface to Elements of Geometrie of Euclid of Megara", verfügbar über die Webseite des Projekts Gutenberg (`http://www.gutenberg.org/ebooks/22062`).
- Als frühneuzeitlicher italienischer Vergleichstext: Dante Alighieri "La Divina Commedia di Dante: Inferno" (`http://www.gutenberg.org/ebooks/997`).
- Als frühneuzeitliche deutsche Vergleichstexte: Philipp Melanchthon "Die Augsburger Confession" (`www.gutenberg.org/ebooks/607`) sowie Paracelsus "Das Buch Pangranum" (Webressource inzwischen inexistent)[13].
- Als frühneuzeitlicher lateinischer Vergleichstext: René Descartes "Meditationes de prima philosophia" (`http://www.gutenberg.org/ebooks/23306`).
- Als altertümlicher lateinischer Vergleichstext: Julius Caesar "De Bello Gallico" (`www.gutenberg.org/ebooks/218`)
- Als P.III-Chiffren wurden der Einfachheit halber die oben genannten Vergleichstexte mit diesem Verfahren verschlüsselt.

Die Texte müssen eingelesen, von Format bzw. Metainformationen getrennt und danach tokenisiert werden.[14] Anschließend erfolgen die unterschiedlichen statistischen Analysen. Die Ergebnisse müssen tabellarisch und graphisch aufbereitet werden, um die Werte komfortabel vergleichen zu können. Da jeweils nur wenige verschiedene Texte analysiert werden, wird auf den Einsatz einer Komponente, welche die Texte anhand ihrer ermittelten Eigenschaften in Cluster eingruppiert, in diesem Teil der Analysen verzichtet. Abbildung 6.5 zeigt den Aufbau eines Experiments, welches die statistischen Eigenschaften verschlüsselter und unverschlüsselter Vergleichstexte berechnet.[15]

12 Dem Leser steht frei, einzelne Sektionen oder andere Transkriptionen auf ihre statistischen Eigenschaften zu analysieren. Dazu müssen lediglich die Werte für die Konfigurationen (vgl. 3.2.3.) des VIAFReaders im Experiment geändert werden. Links zu den verwendeten Experimenten finden sich in Anhang D.

13 Der Text wurde in das BuiltIn-Corpus von Tesla integeriert, so dass er beim Tesla-Download jedem zur Reproduktion der Ergebnisse zur Verfügung steht.

14 Die Ermittlung von Tokens ist notwendig für die Berechnung der Wortlängen- und die Häufigkeitsverteilung sowie der Erzeugung von rein aus Buchstaben bestehenden Texten für die Entropie-, Koinzidenz- und Random-Walk-Messungen.

15 Für den VM-Text wurde fast exakt der gleiche Versuchsaufbau verwendet, wie in der oberen Hälfte des Experiments auf dem Screenshot. Da die Prozessierung des VM-Textes eine abweichende Tokenisierungs-Konfiguration benötigt, konnte er nicht einfach den NaSp-Texten zugeschlagen werden, sondern benötigte ein eigenes Experiment.

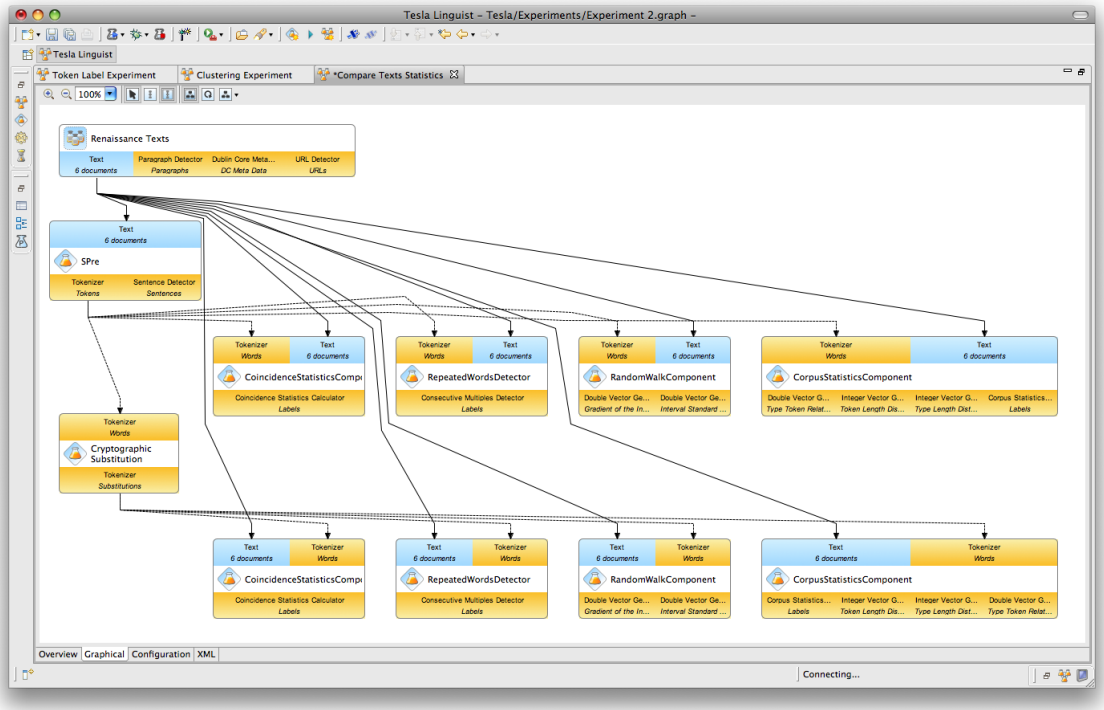


Abbildung 6.1: *Der graphische View auf eines der durchgeführten Experimente: Links oben die Textselektion (es handelt sich um die sechs natürlichsprachlichen Vergleichstexte), darunter als Tokenisierer die Komponente SPre. Text und Tokens sind Input für die vier nebeneinanderliegenden Komponenten (Mitte), welche die zu ermittelnden Eigenschaften der Ausgangstexte berechnen. Die Tokens der Texte werden außerdem mit der Verschlüsselungskomponente (rechter Rand, unten) zu P.III-Texten chiffriert, die dann wiederum Input für die vier Berechnungskomponenten (unten) sind.*

6.1.4 Benötigte Komponenten

Für das experimentelle Design werden eine Reihe von Komponenten herangezogen. **Reader** (vgl. 3.1.3) stellen den Inhalt der an der Analyse beteiligten Texte zur Verfügung. Die Tokenisierung dieser Inhalte übernimmt ein **Präprozessor** (vgl. 3.1.4), der über unterschiedliche Konfigurationen an die verschiedenen Formate angepasst werden kann. Eine **Verschlüsselungskomponente**, welche alle benötigten Verfahren zur Chiffrierung von Texten anbietet, übernimmt die Erzeugung der chiffrierten Vergleichstexte. Schließlich werden unterschiedliche **Statistik-Komponenten** benötigt, um die zu analysierenden Werte der Texte zu berechnen.

Reader

Der Inhalt von Texten wird Tesla durch Reader-Komponenten zur Verfügung gestellt (vgl. 3.1.3). Im Falle IAF, der am besten ausgearbeiteten Ressource für den Text des VM, ist die Bereitstellung eines spezifischen Readers erforderlich, um die Fülle der dort codierten Information innerhalb von Tesla-Experimenten nutzbar zu machen. Im IAF ist der VM-Text in insgesamt 19 verschiedenen Transkriptionen verfügbar,[16] die über den `VoynichInterlinearArchiveReader` mittels der Transcription-Konfigurationseinstellung einzeln extrahiert werden können. Ein Ausschnitt aus dem IAF ist in Listing 6.1 abgebildet. Wie zu sehen, sind sämtliche Transkriptionen im EVA-Alphabet codiert (vgl. 4.2.2). Wünscht man eine Codierung im Currier-Alphabet, so kann man das über die *Currier Alphabet*-Konfigurationsschnittstelle angeben. Über die Bereitstellung der ausgewählten Transkription hinaus liefert der Reader weitere Informationen zu einzelnen Seiten. Diese umfassen eine eindeutige Seitenbezeichnung, die Sektion des VM, zu der die Seite gehört (vgl. 4.2.1) sowie den Currier-Dialekt der Seite (vgl. 4.2.3) und werden als Annotationen mit dem Datenobjekt `IVoynichTextLabel` abgelegt. Mit diesen Analysen kann der Text durch anschließende Komponenten gefiltert werden, um bspw. kontrastive Analysen der Sternen- und der Botanischen Sektion oder der Currier-A-Textteile gegenüber den Currier-B-Textteilen durchzuführen. Der Reader kann alternativ auch direkt in einer Weise konfiguriert werden, um den Text bestimmter Sektionen bzw. einer Currier-Sprache liefern zu lassen. Eine ausführliche Beschreibung dieser und weiterer Funktionalitäten des verwendeten Readers findet sich im Anhang C.1.6; das Format des Voynich IAF ist selbstdokumentierend (die wesentlichen Formatinformationen finden sich innerhalb der ersten 320 Zeilen).

Die für das Experimentdesign notwendigen Vergleichstexte (vgl. 6.1.3) liegen in einem flachen Textformat vor, das nur rudimentär mit Metainformationen angereichert ist (u.a. Kapitelnummern), für den Zugang zum Content wurden leicht spezialisierte Reader eingesetzt, welche die Metainformationen in Annotationen überführen.

16 Darunter finden sich die einzige vollständige Transkription von Takahashi, die nahezu vollständigen von Friedman und Currier genauso wie Transkriptionen, die sich lediglich über einzelne Seiten oder Abschnitte erstrecken.

```
 1 %<f10r>          {$I=H $Q=B $P=C $L=A $H=1}
 2 # Last edited on 1998-10-09 01:33:27 by stolfi
 3 (...)
 4 <f10r.P.11;H>       rotcho.shor.qoty.qotor.cthy.d-{plant}otar-{plant}
 5 <f10r.P.11;C>       rotcho!shor.qoty.qotor.cthy!d-{plant}otar-{plant}
 6 <f10r.P.11;F>       rotcho.shor.qoty.qotor.cthy!d-{plant}otar-{plant}
 7 #
 8 <f10r.P.12;H>       rodaiin.daiin.qotchy.qotor={plant}
 9 <f10r.P.12;C>       rodaiin.daiin.qotchy.qotor={plant}
10 <f10r.P.12;F>       !odaiin.daiin.qotchy.qotor={plant}
```

Listing 6.1: *Ausschnitt aus dem Voynich Interlinear Archive File (IAF). In der ersten Zeile findet sich (hinter der Seitenbezeichnung in spitzen Klammern) die codierte Information, u.a. zur Sektions- und Currier-Sprachen-Zugehörigkeit. Kommentarzeilen beginnen mit #. Textzeilen beginnen mit einem eindeutigen Bezeichner der Form <Seitenbezeichnung.Unitbezeichnung.Zeilennummer;Transkription>. In diesem Fall sind zwei Textzeilen angegeben, die in jeweils drei verschiedenen Transkriptionen vorliegen. Der Reader liefert als Content den String hinter dem Bezeichner mit der ausgewählten Transkription und annotiert die Daten mit den weiteren Informationen.*

Präprozessor

Der vom Reader gelieferte Content des VM ist in einem speziellen Format abgelegt. Die einzelnen Wörter des Textes sind nicht durch Spatien, sondern durch Punkte voneinander getrennt. Zeilenenden werden durch einen horizontalen Strich markiert (-), Paragraphenden durch zwei (=). Die einzelnen Transkriptionen sind im IAF zur besseren Vergleichbarkeit aligniert, dabei symbolisieren Ausrufezeichen (!) Lücken. Hinweise zu Zeichnungen, die in den Text hineinragen, werden innerhalb geschweifter Klammern gegeben.[17] Im folgenden Beispiel (Content der H-Transkription aus Listing 6.1) etwa trennt ein Teil einer Pflanzenzeichnung die erste Zeile in zwei Teilstücke und begrenzt den Text beider Zeilen auf der rechten Seite:

rotcho.shor.qoty.qotor.cthy.d-{plant}otar-{plant}

rodaiin.daiin.qotchy.qotor={plant}

In Kapitel 3.1.4 wurde der Präprozessor `SPre` vorgestellt, der sich über unterschiedliche Konfigurationen beliebigen Textformaten anpassen kann. Zum Zwecke der Tokenisierung des VM wurden Konfigurationen für drei konsekutive Tokenisierungsstufen angelegt: In der VMCharacterParser-Konfiguration werden die validen Zeichen definiert und nach Buchstaben, Trennzeichen und Skopuszeichen klassifiziert. In der VMWordParser-Konfiguration werden Wörter (als Elemente zwischen Punkten) und Kommentare (als

17 Dies gilt auch für weitere Anmerkungen, die sich als inline-Kommentare innerhalb des Contents finden.

zwischen geschweiften Klammern stehend) definiert, der VMParagraphParser schließlich definiert Paragraphen. In Ermangelung weiterer funktionaler Einheiten (die Zeilen wurden ja bereits im Reader als solche annotiert, s.o.) muss der Präprozessor keine weiteren Elemente definieren.

Für die Tokenisierung der Vergleichstexte wird SPre mit dem Unicode-TemplateSet konfiguriert. Dieses definiert sämtliche Buchstaben des Unicode über die `isLetter(char)`-Methode der Java-Klasse `Character` als valide Buchstaben, die Teil von Wörtern sein können. Darüber hinaus werden Abkürzungslisten genutzt, die verhindern, dass Abkürzungspunkte als Satzenden missinterpretiert werden.[18]

Verschlüsselungskomponente

Der P.III-Text muss für die Analysen über das entsprechende Verschlüsselungsverfahren erzeugt werden. Um die Ergebnisse nachvollziehbar und überprüfbar zu halten, muss der Prozess der Verschlüsselung ebenfalls in dokumentierten Experimenten stattfinden. Dies wird erreicht über die Einbindung der `CryptographicSubstitutionComponent`, einer Tesla-Komponente, die in der Lage ist, die erforderlichen Chiffriermethoden anzubieten. Durch Konfiguration mit dem PolygraphiaIIICipherGenerator-Template-Set vermag sie, Texte über die Substitutionslisten der P.III zu verschlüsseln.[19]

Trithemius äußert sich selbst nicht darüber, wie bei der Verschlüsselung anhand der P.III-Ersetzungstabellen konkret vorgegangen werden soll. Die für die Anwendung der PI- und PII-Tabellen vorgesehene konsekutive Folge (der erste Buchstabe wird durch seine Chiffre aus der ersten, der zweite durch dessen Pendant aus der zweiten Spalte ersetzt) beruhte auf dem notwendigen syntaktischen und semantischen Anschluss zwischen den einzelnen Wörtern des steganographischen lateinischen Textes, der möglichst unauffällig erscheinen sollte. Solcherlei syntaktische und semantische Anschlüsse finden sich in einem phantasiesprachlichen Text selbstredend nicht; dem Anwender ist also freigestellt, in welcher Reihenfolge er aus welchen Spalten seine benötigten Ersetzungseinheiten gewinnt.

Für die automatische Generierung eines Textes mit der P.III-Verschlüsselung ergibt sich so die Frage, wie eine plausible Auswahl der Einheiten simuliert werden kann. Relativ unwahrscheinlich ist es, dass ein menschlicher Anwender eine echte Zufallsfolge über die

18 Zum Funktionsumfang von SPre vgl. auch Anhang C.2.1.

19 Darüber hinaus bietet sie noch weitere Methoden an, die für die hier durchgeführten Analysen nicht verwendet werden, u.a. kann monoalphabetisch, polyalphabetisch und über die weiteren von Trithemius beschriebenen Methoden chiffriert werden; vgl. Anhang C.2.2.

zur Verfügung stehenden 132 Ersetzungseinheiten pro Klarbuchstabe nachahmen kann: Dazu müsste dieser sämtliche Ersetzungstabellen im Blick haben und zudem in der Lage sein, diese zufällig auszuwählen. Wahrscheinlicher ist ein Szenario, in dem die Tabellen – wie im Fall der P.III – auf einzelne Seiten eines Blockes oder Buches aufgeteilt sind, also immer nur ein Teil von ihnen zugänglich ist. In der Polygraphia finden sich im P.III-Teil jeweils drei Spalten pro Seite. Bei einem aufgeschlagenen Buch ergeben sich mithin sechs sichtbare Spalten. Aus diesen sechs Spalten können nun Ersetzungseinheiten ausgewählt werden, nach dem Umblättern findet man sechs neue Spalten, die wiederum mehrfach angewendet werden. Dabei muss man nicht immer auf die nächste Seite blättern, sondern evtl. mehrere überschlagen, zwischendurch wieder zurückblättern, bestimmte Seiten exzessiv nutzen oder nach der ersten Ersetzungseinheit schon weiterblättern. Ein solches Verhalten ist zwar relativ schwer durch eine automatisierte Methode nachzubilden, dennoch wurde ein Versuch unternommen, dem mit einem Algorithmus möglichst nahe zu kommen. Das Verfahren wurde durch eine probabilistische *StateMachine*, also durch Zustände und gewichtete Überführungsfunktionen, dargestellt: Zunächst wurde die gesamte Anzahl der Spalten in sechs Einheiten umfassende Cluster unterteilt, die Zuständen entsprechen. Eines der Cluster fungiert als Ausgangszustand, innerhalb dessen die erste Spalte zufällig aus den sechs vorhandenen ausgewählt wird. Nach dem Auswahlprozess besteht eine gewisse Wahrscheinlichkeit, dass der Auswahlprozess im Cluster verbleibt (die nächste Spalte wird auch aus den aufgeschlagenen Seiten ausgewählt) oder zu einem anderen Cluster wechselt (die Seite wird umgeblättert). Zusätzlich lässt sich annehmen, dass die StateMachine nach einer bestimmten Anzahl von Ersetzungsprozeduren in ihren Ausgangszustand zurückkehrt und die Abfolge auf ähnliche Weise wiederholt wird.

Statistikkomponenten

Für die Berechnung der benötigten statistischen Kennwerte werden unterschiedliche Komponenten genutzt. Die `CorpusStatisticsComponent` bietet die Berechnung der gängigen traditionellen korpusstatistischen Werte (Häufigkeitsverteilungen, Entropie) an. Über die Konfiguration lässt sich steuern, ob etwa die Statistik von Einzeltexten oder die von Textsammlungen (Selections) berechnet werden soll. Darüber hinaus können einzelne – mitunter kostenintensive – Berechnungen an- bzw. abgeschaltet werden.

Die Ermittlung der Anzahl der Wortzwillinge und -drillinge wurde in eine eigene Komponente (`RepeatedWordsDetector`) ausgelagert, da eine solche Funktion nicht zu der gängigen korpusstatistischen Analyse gehört. Die Komponente zählt sowohl wirkliche Mehrlinge

(also aufeinander folgende Wörter, deren Buchstabenfolgen sich exakt entsprechen), als auch Minimalpaarmehrlinge (also aufeinander folgende Wörter, die sich in genau einem Buchstaben unterscheiden).

Berechnungen von Koinzidenzwerten (Kappa, Chi, Psi und Phi) werden von der `CoincidenceStatisticsComponent` durchgeführt, bei der eingestellt werden kann, ob sie ihre Werte innerhalb von Texten berechnet (also die erste Hälfte der zweiten Texthälfte gegenüberstellt) oder ob sie die verschiedenen Texte der Korpusselektion paarweise einander gegenüberstellt. Im Falle der hier durchgeführten Analysen werden jeweils die Werte innerhalb der Texte berechnet.

Die Berechung der Intervall-Standardabweichungen für die Detektion von LRCs erfolgt in der `RandomWalkComponent`. Bei der Umsetzung des Random Walk Modells werden dabei zufällige (aber reproduzierbare) Werte für die einzelnen Buchstaben gesetzt. Detaillierte Informationen zu den erwähnten Statistik-Komponenten finden sich im Anhang C.

6.1.5 Ergebnisse

Zunächst werden die für die Texte ermittelten Werte der einzelnen Text-Merkmale, über welche die Haupthypothese aus 6.1.2 operationalisiert wurde, angeführt.[20] Im letzten Abschnitt dieses Kapitels werden die Resultate zu den einzelnen Merkmalen der Texte zusammengetragen, um eine Aussage hinsichtlich dieser Haupthypothese zu treffen.

Type-Token-Relation

Bei der Zählung von Types und Tokens des VM fällt zunächst auf, dass die Anzahl unterschiedlicher Wörter keinesfalls so "surprisingly limited" ist, wie von D'Imperio (1978b:28) behauptet: Insgesamt finden sich über 8000 verschiedene Types im VM, was bei knapp 38000 Tokens eine Gesamt-TTR von 21,48 ergibt. Die Vergleichstexte entsprechender Länge liegen zum Teil beträchtlich unter diesem Wert (siehe Tabelle 6.3). Auch die über Kohorten von 1000 Tokens standardisierte TTR liegt beim VM bei über 50 und ist damit vergleichbar mit den Werten von NaSp-Texten. P.III-verschlüsselte Texte können im Höchstfall 132*24=3168 Types aufweisen,[21] von denen – je nach Auswahlverfahren – unterschiedlich viele genutzt werden. Im oben beschriebenen Auswahlverfahren, das eine

20 Wichtiger als die Ergebnisse ist – wie in dieser Arbeit schon mehrfach erwähnt wurde – die Art ihres Zustandekommens. Deshalb finden sich im Anhang D Verweise zu allen angeführten Experimenten und Komponentenversionen, damit sie von jedem Leser zu jeder Zeit reproduziert werden können.

21 Genaugenommen ist die Zahl unerheblich kleiner, da Trithemius Stämme mehrfach benutzt, siehe A.2.

StateMachine nutzt, werden für unterschiedlich lange Chiffren-Texte zwischen 2100 (gekürzter Melanchton-Text, ca. 10000 Tokens) und 2600 (Paracelsus-Text, fast 100.000 Tokens) Types genutzt. Das Verfahren generiert demnach sehr unterschiedliche Gesamtwerte für die TTR, die pro 1000 Tokens standardisierte Methode ergibt allerdings den relativ hohen Wert von 73.

	Dee	**Descartes**	**Dante**	**Melanch**	**Paracelsus**
Tokens	31679	19318	34246	13202	29716
Types	4963	4261	6460	2482	4025
TTR	16	22	19	19	14
STTR	43	51	51	43	38

	VM Gesamt	**VM CurrA**	**VM CurrB**	**P.III Melanch**	**P.III Short**
Tokens	38026	11461	23221	66731	11264
Types	8141	3441	4942	2434	2170
TTR	21	30	21	4	19
STTR	54	54	51	73	74

Tabelle 6.3: *Aufstellung der Werte für die Zählung von Tokens und Types in den analysierten Texten, sowie der Type-Token-Relation in den Gesamttexten (Zeile* TTR*) und der Mittelwert über Kohorten von 1000 Tokens (Zeile* STTR*). Oben: Natürlichsprachliche Vergleichstexte, unten: VM Gesamttext, Seiten des VM in Currier A, Seiten des VM in Currier B, Melanchthons Confessions (deutsch, Gesamttext) P.III-chiffriert sowie der Anfang des gleichen Textes, mit dem gleichen Verfahren chiffriert (P.III Short).*

Gegen die Haupthypothese, hinter dem VM-Text stände eine P.III-artige Chiffre, spricht zweifellos die hohe Anzahl unterschiedlicher Types im VM. Selbst wenn man für die beiden Currier-Sprachen unterschiedliche Ersetzungstabellen annehmen würde, bräuchte man – nimmt man an, dass 24 Buchstaben verschlüsselt werden – mindestens 150 (Currier A) bzw. mehr als 200 (Currier B) Spalten in der Ersetzungstabelle. Es wurde allerdings bereits erwähnt, dass viele der VM-Wörter singulär vorkommen, und nur an ausgewiesenen Stellen platziert sind, z.B. als Pflanzen-Label oder als eine Art Füllwort am Zeilenende. Interessant für die nachfolgenden Betrachtungen, insbesondere für die Bemühungen zur Schlüsselrekonstruktion in 6.2, wäre möglicherweise ein Ausschluss dieser *hapax legomena*, die zumindest zum Teil auch durch einfache Schreib- oder Lesefehler überhaupt erst auftreten.[22]

22 Tatsächlich ist die Anzahl der hapax legomena im VM mit 70% der Types signifikant höher als jene der NaSp-Vergleichstexte (zwischen 50 und 60%). Zu beachten ist auch, dass die P.III-Chiffre durch

Wortlängen

Ergebnisse, die recht eindeutige Ähnlichkeiten zwischen VM und P.III-Chiffre zutage fördern, ergeben sich aus den Analysen hinsichtlich der Wortlängenverteilung: Während Types natürlichsprachlicher Texte den höchsten Punkt in ihrer Verteilung (Peak) bei 6 und das arithmetische Mittel bei 6,5-8 Zeichen pro Wort haben, weist ihre Verteilung eine signifikant größere Breite auf als die von VM- und P.III-Text. Noch deutlicher zeigt sich das bei Zählung der Token-Wortlängen: Während sich die Wortlängenverteilungen für Types und Tokens P.III und VM kaum unterscheiden, treten kurze Wörter in NaSp-Texten sehr viel frequenter auf als längere. Die Verteilung der Token-Wortlängen hat deshalb ihren Peak sehr viel früher, die Verteilung ist insgesamt wieder breiter als die der Chiffrentexte. Auf die Breite der Verteilung lässt sich über die Standardabweichung der Längen von Wörtern schließen (siehe Tabelle 6.4, Zeilen 2 und 4): Sowohl die Längen von Tokens, als auch die von Types streuen bei den NaSp-Texten signifikant stärker als die entsprechenden Werte im VM-Text. Aus den entsprechenden Werten für die P.III-Chiffren lässt sich schließen, dass deren Wörter hinsichtlich ihrer Länge noch weniger als die des VM streuen.[23]

	Dee	**Descartes**	**Dante**	**Melanch**	**Paracelsus**
MW Tokens	4,69	5,36	3,93	5,38	4,68
SD Tokens	2,68	2,69	2,23	2,73	2,33
MW Types	7,26	7,68	6,45	7,78	7,46
SD Types	2,51	2,51	1,92	2,86	2,62

	VM Gesamt	**VM CurrA**	**VM CurrB**	**P.III Mel.**	**P.III Short**
MW Tokens	5,05	4,87	5,15	5,49	5,49
SD Tokens	1,76	1,76	1,70	1,08	1,08
MW Types	6,36	6,04	6,26	5,59	5,59
SD Types	1,75	1,73	1,76	1,08	1,08

Tabelle 6.4: *Aufstellung der Werte für die durchschnittlichen Wortlängen (MW) und die Standardabweichungen (SD), jeweils für Tokens und für Types. Texte wie in 6.3.*

Werden die Vorkommen der einzelnen Wortlängen in einem Diagramm visualisiert, so

ein rein maschinelles Verfahren generiert wurde, Schreib- und Interpretationsfehler also ausgeschlossen sind.

23 In der Currier-Alphabet-Transkription sind die Werte von VM (MW Types 5,4, SD Types 1,5) und P.III noch näher beieinander.

lässt sich – vor allem bei den Token-Werten – unschwer erkennen, dass Wortlängen im VM-Text und in der P.III-Chiffre relativ normalverteilt sind, während dies für die Wortlängenverteilung der NaSp-Vergleichstexte nicht gilt (siehe 6.2).

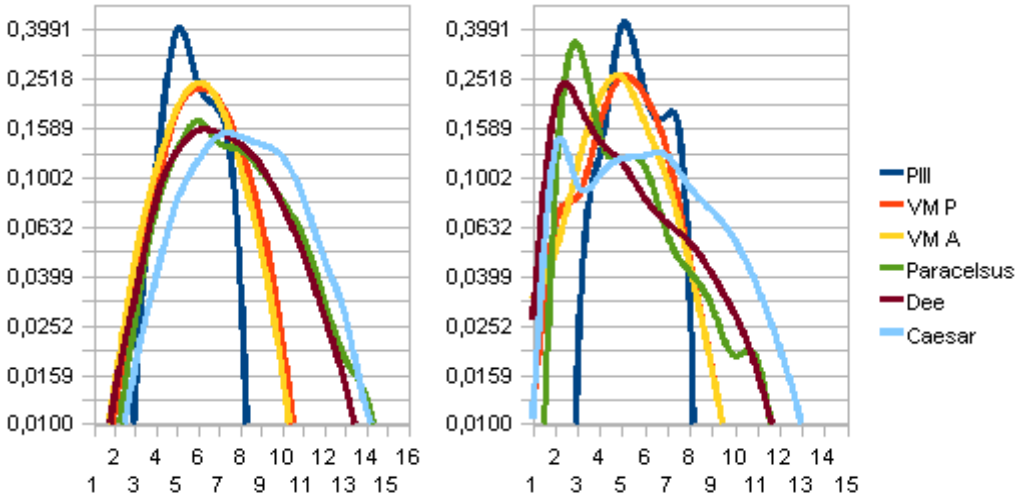


Abbildung 6.2: *Relative Häufigkeiten (y-Achse) der Wortlängen (x-Achse) von Types (links) und Tokens (rechts) ausgewählter Texte. Auffällig ist die strikte Binomialverteilung der VM-Wortlängen, die von der P.III-Chiffre etwas ungleichmäßiger und noch weniger streuend nachempfunden wird. Weiterhin ist gut zu erkennen, dass die NaSp-Texte allesamt sehr breit streuen und – im Gegensatz zu VM und P.III – völlig unterschiedliche Verteilungen für Types und Tokens aufweisen.*

Entropie

Wie von Zandbergen (2011c) behauptet, weicht die Wortentropie im VM-Text nicht signifikant ab von den Werten, die für natürliche Sprachen festgestellt werden (vgl. Tabelle 6.5, Zeile 1). Zandbergen führt das darauf zurück, dass die Zeichenentropie innerhalb von VM-Wörtern gleichmäßiger verteilt ist, als das in NaSp-Wörtern der Fall ist.[24]

Auffällig ist das VM vor allem aufgrund seiner niedrigen Zeichenentropie, was weniger die Entropie von Einzelzeichen (ausgedrückt im Wert H1, Tabelle 6.5, Zeile 3) als die von Zeichenkombinationen (Wert H2, Tabelle 6.5, Zeile 4) betrifft. Bennett (1976) erwähnte die Ähnlichkeit zu H2-Werten polynesischer Sprachen, die aufgrund der Herkunft des VM

24 Leider war es nicht möglich, die Werte, die Zandbergen für diese *zeichenweise Entropie* ermittelt, zu reproduzieren, evtl. weil das Verfahren nicht ordnungsgemäß implementiert wurde. Die entsprechende Methode findet sich in der `tesla.compoment.CorpusStatistics`-Komponente in der Klasse `EntropyCalculator`, falls der Leser das Verfahren in Tesla überprüfen möchte.

	Dee	**Descartes**	**Dante**	**Melanch**	**Paracelsus**
Wörter	9,27	9,94	9,66	9,14	8,93
H0	5,36	4,70	4,59	4,76	4,95
H1	4,10	4,01	3,93	4,03	4,04
H2	3,29	3,28	3,11	3,12	3,14
H1-H2	0,81	0,73	0,82	0,91	0,90

	VM Gesamt	**VM CurrA**	**VM CurrB**	**P.III Mel.**	**P.III Small**
Wörter	10,46	9,88	9,89	10,74	6,52
H0	4,59	4,46	4,46	4,52	4,00
H1	3,88	3,85	3,88	4,00	3,57
H2	2,16	2,17	2,01	2,90	1,90
H1-H2	1,72	1,68	1,87	1,10	1,67

Tabelle 6.5: *Aufstellung der Werte für Wortentropie und die drei zeichenbasierten Entropien (H0 entspricht H_{max}, H1 der Zeichenentropie, H2 der Verbundentropie). Die letzte Zeile gibt die nach Stallings (1998) relevante Differenz von H1 und H2 wieder. Texte wie in Tabelle 6.3, außer dass* P.III Small *hier von einem P.III-Verfahren mit nur 10 (von 132) Substitutionsspalten generiert wurde.*

doch als eher unwahrscheinliche Kandidaten als Grundlage des Manuskripts erschienen. Stallings (1998) zufolge ist die Differenz von H1 und H2 aussagekräftiger, als H2 für sich alleine betrachtet (vgl. 4.2.3). Er entwickelte sogar eine Chiffre, die ähnliche Werte erzeugte ("Verbose Cipher"), indem sie die Hälfte der Klartextbuchstaben durch einzelne Zeichen, die andere Hälfte durch redundante Zeichenfolgen (etwa *bqbababa* für b) ersetzte. Stallings gibt selbst zu, dass die erzeugte Chiffre sich im Erscheinungsbild deutlich vom VM-Text unterscheidet. Betrachtet man nun die letzte Zeile aus Tabelle 6.5, so fällt auf, dass die H1/H2-Differenz für das VM deutlich höher liegt, als die für die NaSp-Vergleichstexte. Der entsprechende Wert für die P.III-Chiffre liegt bei der Nutzung sämtlicher 132 Substitutionsspalten zwar über dem Wert für NaSp, aber noch immer deutlich unter dem für das VM.[25] Wählt man allerdings zufällige 10 Spalten aus, so ergibt sich für H1-H2 ein dem VM-Wert vergleichbarer Wert (letzte Spalte). Und dieser Wert wird von einer Chiffre produziert, die dem VM-Text vom Erscheinungsbild her sehr ähnlich ist.[26]

25 Bei Verwendung des Currier-Alphabets erzielt man wiederum eine Annäherung: H1 liegt bei 3,83, H2 bei 2,39, H1-H2 liegt also bei 1,44. H0 liegt beim Currier-Alphabet allerdings wesentlich höher (5,21), da das Alphabet umfangreicher ist (38 Zeichen).

26 Es ist sicherlich einzuwenden, dass mit der abweichenden Verwendung der Chiffre die Anzahl von Types sehr klein wird (in dem betreffenden Text finden sich nur 129) und dass andere Werte – wie z.B. die Wortentropie – stark vom VM-Wert abweichen. Dieser Einwand wird weiter unten diskutiert.

Koinzidenzwerte

Bei Betrachtung der Koinzidenzwerte lassen sich Ähnlichkeiten zwischen dem VM-Text und den deutschen Vergleichstexten sowie der P.III-Chiffre ausmachen. Alle Werte liegen bei 0,074 oder knapp darüber. Die Differenz zum englischen Vergleichstext ist relativ stark, zum lateinischen und italienischen nur schwach ausgeprägt. Einen Ausreißer nach oben weisen die Werte für den CurrierA-Teil des VM auf (Tabelle 6.6, Spalte 2 unten), die signifikant erhöht sind. Wie eine solche Erhöhung nachgebildet werden kann, zeigt die Anwendung der P.III-Chiffre mit weniger Ersetzungsspalten (Spalte 5 unten).[27]

	Dee	**Descartes**	**Dante**	**Melanch**	**Paracelsus**
Kappa	0,068	0,072	0,073	0,075	0,077
Chi	0,067	0,071	0,072	0,077	0,076
Psi	0,067	0,072	0,072	0,077	0,076
Phi	0,067	0,072	0,072	0,077	0,076

	VM Gesamt	**VM CurrA**	**VM CurrB**	**P.III Mel.**	**P.III Small**
Kappa	0,075	0,077	0,074	0,074	0,086
Chi	0,076	0,080	0,076	0,073	0,086
Psi	0,077	0,083	0,077	0,073	0,086
Phi	0,077	0,083	0,077	0,073	0,085

Tabelle 6.6: *Aufstellung der Koinzidenzwerte. Texte wie in 6.5, P.III Small verwendete 24 Ersetzungsspalten.*

Minimalpaare

Ein großes Rätsel des VM war das oft verwendete Muster von Wiederholungen des gleichen bzw. eines ähnlichen Wortes. Wie in Tabelle 6.7 zu sehen, sind solche Wiederholungen zumindest in einer derartig hohen Frequenz in natürlichen Sprachen eher unüblich.[28] Die P.III-Chiffre liefert – wendet man sie mit der oben beschriebenen StateMachine an – leicht erhöhte Werte, die aber noch nicht an die des VMs herankommen. Die abermalige Anwendung der P.III mit nur 10 Ersetzungsspalten aber zeigt, dass man die Werte des VM auch mit einer maschinellen Methode bei weitem übertreffen kann.

27 Werte für VM mit Currier-Alphabet: Kappa 0,085; Chi, Psi und Phi 0,086.

28 Christoph Benden wies mich darauf hin, dass es eine Reihe von Sprachen (Indonesisch, Javanisch) gibt, die partielle und sogar volle Reduplikationen z.B. zur Pluralbildung verwenden.

	Dee	**Descartes**	**Dante**	**Melanch**	**Paracelsus**
MP2	5,53	8,07	2,61	4,30	6,40
MP3	0,08	0,17	0,10	0	0
WH2	0,98	1,17	0	0,16	2,56
WH3	0	0	0	0	0

	VM Gesamt	**VM CurrA**	**VM CurrB**	**P.III Mel.**	**P.III Small**
MP2	31,47	35,32	30,25	11,12	222,23
MP3	1,89	2,26	1,70	0,06	47,21
WH2	8,51	9,59	8,46	1,36	25,65
WH3	0,30	0,27	0,35	0	0,54

Tabelle 6.7: *Zählung von adjazenten Minimalpaaren (MP2) und Minimaltriplets (MP3) sowie zwei (WH2) und drei (WH3) direkt aufeinanderfolgenden Wortwiederholungen. Texte wie in 6.5, P.III Small verwendete 10 Ersetzungsspalten.*

Random Walk

Experimentell bestätigt werden konnte, dass die Steigung der Kurve über die Standardabweichungen bei NaSp-Texten in der Nähe vom Wert 0,5 bleibt und die Werte für das VM signifikant höher liegen. Dabei konnte aber weder der von Schinner postulierte Wert von 0,85 noch deren plötzliches Auftreten nach 1-3 Textzeilen reproduziert werden, jedenfalls nicht, wenn man nur eine Transkription und nicht das gesamte EVA-File als Text für den Random Walk auswählt. Vielmehr wurde ein gleichmäßiger Anstieg der Steigung festgestellt. Aber auch der leicht erhöhte Wert des VM konnte durch unterschiedliche Anwendungen der P.III-Methode nicht nachgeahmt werden. Möglicherweise ist dies aber durch eine abweichende Auswahlmethode möglich (vgl. Diskussion).

	Dee	**Descartes**	**Dante**	**Melanch**	**Paracelsus**
alpha	0,51	0,53	0,52	0,51	0,51

	VM Gesamt	**VM CurrA**	**VM CurrB**	**P.III Mel.**	**P.III Small**
alpha	0,60	0,54	0,58	0,47	0,49

Tabelle 6.8: *Steigung der Standardabweichungskurve bei 10% der Textlänge.*

Diskussion

Zusammengefasst führte die Überprüfung der einzelnen in 6.1.2 aufgestellten Teil-Hypothesen zu folgenden Ergebnissen:

1. Type-Token-Relation: Weder VM- noch P.III-Text weisen im Vergleich zu NaSp-Texten signifikant abweichende Werte auf. Die Anzahl der Types im VM (>8000) erscheint hoch für eine Codebuch-Chiffre, allerdings bleiben nach Abzug der hapax legomena (die möglicherweise frei erfundene Füllwörter ohne Codebucheintrag, also Nullen oder bloße Schreibfehler sein könnten) nur 2513 Types übrig. Dieser Wert ist mit dem der experimentell erzeugten P.III-Chiffren vergleichbar, die aus 132 Ersetzungsspalten gewonnen wurden.
2. Wortlängen: VM- wie auch P.III-Text verhalten sich hinsichtlich der Kenndaten für Wortlängen (Mittelwert, Standardabweichung, Typelänge vs. Tokenlänge, Normalverteilung) ausgesprochen ähnlich. Die P.III-Wörter sind insgesamt eingeschränkter in der Längendifferenzierung als die VM-Wörter; im Vergleich zu den Wörtern der natürlichsprachlichen Vergleichstexte fallen sie denoch definitiv in die gleiche Klasse.
3. Wortentropie: Keine signifikanten Unterschiede von P.III-, VM- und NaSp-Texten.
4. Verbundentropie: Das experimentell angewendete Auswahlverfahren über sämtliche Codespalten der P.III erzeugte zwar einen signifikant niedrigeren H2-Wert als der entsprechende Wert der natürlichsprachlichen Vergleichstexte, dieser lag jedoch auch signifikant über dem H2-Wert des VM-Textes. Das gleiche gilt für die Differenz aus H1 und H2. Es konnte jedoch gezeigt werden, dass die Reduktion der Anzahl der bei der Chiffrenerzeugung eingesetzten Codespalten dem VM-Text vergleichbare Werte erzeugen kann.
5. Koinzidenzwerte: VM- und P.III-Text liefern dem frühneuzeitlichen Deutsch ähnliche Werte, die sich allerdings auch nicht signifikant von den Werten für das frühneuzeitliche Italienische und das Lateinische unterscheiden. Starke Unterschiede gibt es zum frühneuzeitlichen Englisch.
6. Anzahl der Minimalpaare (Wörter, die sich nur in einem Zeichen unterscheiden): Bei VM und P.III signifikant häufiger als bei NaSp, bei der P.III können durch Modifikationen des Auswahlverfahrens beinahe beliebig hohe Werte erzielt werden.
7. Wortzwillinge/drillinge: Bei VM und P.III signifikant häufiger als bei NaSp, bei der P.III können durch Modifikationen des Auswahlverfahrens wiederum beinahe beliebig hohe Werte erzielt werden.

8. Weitreichende Korrelationen (LRC): Die experimentell ermittelte Steigung der auf einer log-log-Skala aufgetragenen Intervall-Standardabweichungen des VM-Textes erreichte nicht den Wert, der in der Literatur angegeben wurde. Gleichwohl war der VM-Wert signifikant höher als der Wert für NaSp, was durch die Anwendung der P.III-Chiffre nicht nachempfunden werden konnte.

Diese Ergebnisse wirken sich auf die Haupthypothese, welche die Ähnlichkeit zwischen VM- und P.III-Text behauptete, auf unterschiedliche Art aus: Die ermittelten Werte aus 1,3 und 5 belegen zwar die Ähnlichkeit von VM-Text und P.III-Chiffre. Da sie sich aber nicht wesentlich von den ermittelten Werten derselben Merkmale für die natürlichsprachlichen Vergleichstexte unterscheiden, können sie nicht für eine Bestätigung der Haupthypothese verwendet werden. Auffällige Ähnlichkeiten bezüglich der inneren Struktur von VM- und P.III-Wörtern sowie deren Kombinationen zeigen die Ergebnisse aus 2, 4, 6 und 7. Vor allem werden hier auch die Unterschiede zwischen diesen Chiffren und natürlichsprachlichen Texten deutlich. Lediglich die LRC-Analyse (8) liefert keine Ergebnisse, welche die Hypothese untermauern. Möglicherweise ließe sich dies durch ein andersartiges Auswahlverfahren über die Ersetzungseinheiten der P.III beheben.

Auch wenn anhand dieser Ergebnisse nicht alle Zweifel ausgeräumt werden können, dass hinter dem VM-Text eine Chiffriermethode steckt, die nach Art der von Trithemius erdachten Polygraphia-III-Methode funktioniert, so deuten die durchgeführten statistischen Analysen doch darauf hin, dass zumindest die Möglichkeit besteht. Zugegebenermaßen und verständlicherweise ist es nicht der Fall, dass man die trithemischen Substitutionstabellen unmittelbar in den Zeichensatz des VM transferieren und dadurch einen Text erzeugen kann, der dem VM in allen Merkmalen entspricht. Gewisse Unterschiede in den Entropiewerten deuten darauf hin, dass die Regeln für die Kombination von Zeichen im VM noch restriktiver sind als in der P.III-Chiffre. Dies ließe sich aber durch selbst hergestellte Substitutionstabellen, welche diese Restriktionen beachten, eventuell beheben. Eine weitere Herausforderung stellt das Auswahlverfahren der Chiffren dar. Aufgrund der Nachvollziehbarkeit, die im Kontext dieser Arbeit ein essentieller Bestandteil ist, wurden lediglich maschinell erzeugte Auswahlverfahren genutzt, die offensichtlich nicht adäquat das wie auch immer geartete Auswahlverfahren des VM-Skriptes nachbilden konnten. Interessant wäre hier die manuelle Zusammenstellung einer Chiffre mit VM-ähnlichen Wörtern.

Hier wird die Meinung vertreten, dass die Indizien die Annahme unterstützen, der Text des VM sei das Produkt einer Klartext-Chiffrierung mit einer P.III-ähnlichen Methode.

Ein Großteil der *oddness*, welche von den verschiedensten Autoren dem Verhalten des VM-Textes immer wieder zugeschrieben wurde, wird durch ein derartiges Verfahren erklärt. Für diejenigen Merkmale, in denen sich das VM weiterhin von den generierten P.III-Chiffren unterscheidet, besteht zumindest die Möglichkeit, dass diese Unterschiede durch Modifikationen des Auswahlprozesses und/oder der Chiffrengestalt beseitigt werden können.

Hier wird vorerst davon abgesehen, eigene Substitutionstabellen händisch zu generieren, bei denen die Wortlänge zwar normalverteilt, aber etwas breiter angelegt und deren Bigramm-Verteilung eingeschränkter als die der P.III-Wörter ist. Auch die Erprobung weiterer Auswahlverfahren, welche vorhersehbarere Wortfolgen produzieren und damit die Random-Walk-Werte des VM reprodzieren, wird auf weiterführende Forschungen verschoben, weil im Kontext dieser Arbeit der Einsatz des Systems Tesla für die Textprozessierung im Vordergrund stehen soll. Um zu zeigen, dass Tesla nicht nur zur Tokenisierung, Chiffrierung und Berechung korpusstatistischer Kennwerte genutzt werden kann, wird im nächsten Kapitel die Fragestellung modifiziert, indem Verfahren zur Strukturaufdeckung in Texten thematisiert und innerhalb neuer Komponenten implementiert werden.

6.2 Die Suche nach dem Schlüssel

Als Prämisse für die weiteren Überlegungen nehmen wir an, der Text des Voynich-Manuskripts wäre tatsächlich entstanden, indem der Produzent jeden Buchstaben eines Klartextes durch ein VM-Wort aus einem ihm vorliegenden Codebuch ersetzt hätte. Selbstverständlich ist man – vor allem zum gegenwärtigen Zeitpunkt – weit davon entfernt, zu beweisen, dass es exakt auf diese Weise vor sich gegangen ist. Viele der Ähnlichkeiten zwischen dem Text des VM und der produzierten P.III-Chiffre deuten indes darauf hin, dass es zumindest im Bereich des Möglichen liegt, dass beide Texte durch vergleichbare Verfahren generiert wurden.

Der Schlüssel zu einem solchen Verfahren ist das Codebuch, in dem die einzelnen Chiffren zur Ersetzung der Klartextbuchstaben eingetragen sind. Ein derartiges Codebuch ist im Fall des VM nicht verfügbar.[29] Man kann wohl auch nicht darauf hoffen, dass es irgendwo in einem Archiv versteckt ist und demnächst entdeckt wird. Bei der Analyse der unterschiedlichen Codebuch-basierten trithemischen Chiffren der Polygraphia (Kapitel 5.3.3)

29 Amüsanterweise verhält es sich im Fall der P.III genau umgekehrt, da lediglich das Codebuch, aber kein damit erzeugter Text vorliegt.

wurde festgestellt, dass die Erfolgsaussichten kryptoanalytischer Angriffe zwischen den beschriebenen Verfahren stark divergieren: Während sich die Codebücher für die Ave-Maria-Chiffre (Polygraphia I & II) wahrscheinlich gar nicht rekonstruieren ließen, könnte ein solches Vorhaben für die auf Phantasiewörtern basierende P.III Erfolg haben.

Auf Basis dieser Überlegungen werden in den beiden folgenden Abschnitten Ansätze vorgestellt, die dabei helfen könnten, ein solches Rekonstruktionsvorhaben aus dem Text des VM umzusetzen. Wie im ersten Teil der Analysen (6.1) werden auch hier wieder Verfahren für kryptoanalytische Zwecke eingesetzt, die ursprünglich für ein anderes wissenschaftliches Gebiet – die strukturalistische Linguistik – entwickelt wurden und die in das Framework Tesla integriert sind.

6.2.1 Problemstellung: Rekonstruktion der Substitutionstabellen

Die P.III-Chiffren sind sehr zahlreich (132 Homophone für jeden zu verschlüsselnden Buchstaben) und in Tabellen aufgetragen, denen teils systematische, teils stochastische Strukturen zugrunde liegen. Möglicherweise können wir uns die Tatsache zunutze machen, dass es wahrscheinlich ist, bei der Generierung einer ganzen Reihe von Spalten auf immer die gleichen Muster zurück zu greifen, sei es aus Bequemlichkeit oder Gewissenhaftigkeit. Trithemius verfiel in solche Muster, so vielleicht auch der oder die Autor(en) des VM. Die Aussicht auf Erfolg mag nicht besonders groß erscheinen, doch die in 6.1 angeführten Gründe für die Isomorphie von P.III- und VM-Chiffre geben hier zumindest einen theoretischen Rahmen, der den Versuch, derartige Muster zu finden, nicht völlig abwegig erscheinen lässt. Eine Besonderheit des P.III-Verfahrens ist, dass ähnliche Wörter in allen Fällen Unterschiedliches codieren (vgl. Anhang A: Verschiedene Wörter, die mit dem gleichen Wortstamm beginnen, codieren in jedem Fall unterschiedliche Buchstaben). Übertragen auf das VM liefert dieses Paradoxon möglicherweise eine Erklärung, weshalb die bisherigen Entschlüsselungsversuche durchweg gescheitert sind.

Hier sollen nun Überlegungen angestellt werden, ob und wie Muster im Inventar des VM detektiert werden können, die es in einem weiteren Schritt erlauben, eine mögliche zugrunde liegende Verschlüsselungstabelle zu rekonstruieren. In Abwesenheit einer weiteren Vergleichschiffre werden die in diesem Zuge entwickelten Aufdeckungsverfahren an einer P.III-Chiffre erprobt, um dann auf den VM-Text angewendet zu werden.

Aufdeckungsverfahren, die allein auf den Elementen des Sprachsystems operieren, wurden für die Definition natürlichsprachlicher Einheiten innerhalb der strukturalistischen

Linguistik entwickelt. Der Strukturalismus ist ein geisteswissenschaftlicher Ansatz der europäischen (später auch amerikanischen) Moderne, der sich zum einen auf die sprachtheoretischen Konzepte von de Saussure, zum anderen auf unterschiedliche, teilweise miteinander in Konflikt stehende geisteswissenschaftliche Schulen Anfang bis Mitte des 20. Jahrhunderts bezieht. Als wichtigste dieser Schulen werden die *Prager funktionale Schule*, die *Kopenhagener Glossematik* und der *amerikanische Distributionalismus* angesehen (vgl. Kucharczik 2009). Als vorherrschendes Paradigma der Sprachwissenschaft wurde der Strukturalismus von der Generativen Grammatik verdrängt. Deren Beginn wird meist mit dem Erscheinen von Chomskys *Syntactic Structures* 1957 gleichgesetzt. Sie positionierte sich zwar explizit gegen zu dem Zeitpunkt vorherrschende strukturalistische Ansätze (vor allem den Behaviorismus), setzt allerdings zu großen Teilen auf die im Strukturalismus erarbeiteten Konzepte und Methoden auf und entwickelt diese weiter. Gerade in jüngerer Zeit lassen sich wieder vermehrt computerlinguistisch motivierte Umsetzungen explizit strukturalistischer Ansätze beobachten (vgl. Kapitel 2.1).

Dem Strukturalismus liegt ein deskriptiver Ansatz zugrunde, bei dem Sprache als System angesehen wird,[30] in dem sich die einzelnen Bestandteile durch ihre Beziehungen zueinander erst konstituieren. Diese Beziehungen werden auch als Strukturen bezeichnet.[31] Das sprachliche System wird damit als ein Netzwerk von Relationsbeziehungen gedacht, das heißt als ein System von Werten, die zueinander in Opposition stehen. Die sprachlichen Elemente werden darin innersystemisch über ihre Oppositions-Relationen, also kontrastive Beziehungen zueinander konstituiert, ein Rückgriff auf außersprachliche Eigenschaften ist damit unnötig (vgl. Kucharczik 2009:681). Der Analyse des VM-Textes kommt eine solche Arbeitsweise, die sprach- bzw. hier textintern operiert, sehr entgegen, da im Fall des VM nichts weiteres als der (unverständliche) Text vorliegt. Infolgedessen können all die Argumente, mit denen der Strukturalismus aufgrund seiner allzu strengen Regelhaftigkeit angegriffen wurde (vgl. Albrecht 2002), gegen die hier durchgeführte Anwendung nicht vorgebracht werden, weil hier alleinig nach diesen Regeln gesucht wird.

Beim P.III-Verfahren sind Wörter die größte Sinneinheit, da sie beliebig aus mehreren Tabellen gewählt werden können. Die Teilbereiche der strukturalistischen Linguistik, die

30 Hier soll angemerkt sein, dass sich der Strukturalismus nicht auf sprachwissenschaftliche Schulen beschränkte. In seinem weiter gefassten Ansatz wird nicht nur *Sprache,* sondern *Kultur* als System begriffen.

31 Der Begriff *Struktur* wird dabei von Saussure selbst nicht gebraucht (vgl. Kucharczik 2009:683). Spätere Definitionen bezeichnen Struktur als nach außen abgeschlossene Einheit, die mehr ist, als die Summe ihrer Teile – und zwar die Summe ihrer Teile zuzüglich der Summe der Relationen zwischen den Teilen.

sich mit linguistischen Einheiten unterhalb der Wortgrenze beschäftigen, sind die Phonologie, bzw. – da es sich hier um schriftsprachliche Texte handelt – die Graphemik und die Morphologie. Die nächsten beiden Kapitel werden ausloten, inwieweit die in diesen beiden sprachwissenschaftlichen Teilgebieten entwickelte Methodik beim hier vorgesehenen Anwendungsfall, der Rekonstruktion von Schlüsseltabellen für den VM-Text, hilfreich eingesetzt werden kann.

Klassifikation von Glyphen: Die graphemische Methode

Bei der näheren Betrachtung der P.III-Chiffre fällt auf, dass sich die Buchstaben-substituierenden Wörter in drei Teile gliedern lassen. Dabei spielt der Unterschied zwischen Vokalen und Konsonanten eine große Rolle (siehe 5.3.2). Zur Erinnerung: Der letzte Vokal von P.III-Wörtern grenzt die Menge von Buchstaben ein, die durch das jeweilige Wort verschlüsselt werden können. Endet das Wort beispielsweise mit *a*, gefolgt von einem Konsonanten, so kommen als Klartextbuchstaben nur *a, f, l, q, x* infrage. Diese "Vokalregel" gilt für sämliche P.III-Chiffren. Die weiteren Regelmäßigkeiten (Stammkonstanz, Schlusskonsonanten, vgl. Anhang A) tragen weniger Information, dennoch würde ihre Analyse einen Beitrag für das Rekonstruktionsvorhaben der Substitutionstabellen leisten. Im unbekannten Schriftsystem des VM ist es nicht möglich, Vokale und Konsonanten auszumachen. Da es auf völlig andere Art entworfen sein könnte, würde dieser Unterschied möglicherweise auch keine Bedeutung für die Anordnung in der Tabelle haben. Die hier aufgeführte Unterscheidung zwischen Vokalen und Konsonanten soll nur verdeutlichen, dass sich Zeichen offensichtlich klassifizieren lassen und dass diese Klassenzugehörigkeiten an Muster gekoppelt sind. Vor der Klassifizierung muss Klarheit darüber herrschen, welche Zeichen überhaupt zu klassifizieren sind. Dies mag für Schriftstücke auf den ersten Blick einfacher festzustellen sein als für mündliche Äußerungen, da – wie im ersten Kapitel dargelegt – Texte aus diskreten Einheiten bestehen. Wie allerdings in 4.2.2 gezeigt wurde, ist die Analyse der VM-Glyphen keine elementare Aufgabe – man denke an die verschiedenen Transkriptionen auf Basis unterschiedlicher Transkriptionsalphabete.

Die Analyse sprachlicher Äußerungen ist einer der ältesten Forschungszweige der modernen (im Sinne von post-Saussure'scher) Linguistik. Eines der zentralen Paradigmen dieser Anschauung ist das *Prinzip der doppelten Artikulation*. Eine systematische Darstellung dieses Konzepts liefert Bloomfield (1926): Alle sprachlichen Äußerungen sind Folgen vom Morphemen (den kleinsten bedeutungstragenden Einheiten einer Sprache); Morpheme sind wiederum Folgen von Phonemen (den kleinsten bedeutungsunterscheidenden Einhei-

ten einer Sprache, die selbst keine Bedeutung tragen). Das Inventar an Phonemen einer Sprache wird über die Analyse von Minimalpaaren ermittelt. Ein Minimalpaar besteht aus zwei bedeutungstragenden Einheiten, die sich in genau einem Laut[32] unterscheiden. Das Minimalpaar [rɑt] vs. [tɑt] etwa liefert die Phoneme *r* und *t*. Mittels dieser Methode lassen sich bspw. für das Deutsche etwa 40 Phoneme identifizieren.[33]

Bei Feststellung des Grapheminventars lässt sich analog verfahren, indem nämlich Minimalpaare für schriftlich fixierte Wörter ermittelt werden.[34] Das Inventar, welches Günther (1988:85) für das Deutsche ermittelt, besteht zum überwiegenden Teil aus einzelnen Glyphen, teilweise bilden aber auch Buchstabenkombinationen Grapheme (<ch>, <qu> und <sch>).

Die bloße Ermittlung der Grapheme hilft beim Rekonstruktionvorhaben noch nicht recht weiter. Die Grapheme müssen klassifiziert werden, um die Systematik ihrer Verwendung verstehen und evtl. nutzen zu können. Grundlage jeder Klassifizierung ist die Ermittlung von Merkmalen, auf Basis derer klassifiziert wird. Der hier entworfene Ansatz stützt sich auf die Auswertung von Kontexten, in denen Grapheme als Minimalpaar-unterscheidende Einheiten (MPuE) vorkommen. Dabei können drei Merkmale operationalisiert werden:

1. Minimalpaarpartner (MPP): Grapheme, die das entsprechende Element des komplementären Wortes bilden.
2. Graphemkontext: Unmittelbare Umgebung der MPuE im Wort.
3. Wortposition: Position der MPuE relativ zu Wortanfang und -ende.

Um die Gewinnung der Merkmale an einem konkreten Beispiel zu demonstrieren, betrachte man die beiden P.III-Chiffren *pasun* und *pasan*. Das Minimalpaar definiert gemäß dem Distinktivitätskonzept *u* und *a* als MPP-Grapheme (*a* ist der MPP von *u* und umgekehrt). Die MPuE findet sich im Kontext *{s_ n}*, an der vierten Position (von vorne gezählt) bzw. an der zweiten Position (von hinten gezählt). Auf diese Weise werden die aufgezählten Eigenschaften sämtlicher Minimalpaare des Textes zusammengetragen und

32 Laut wird hier im Sinne von *Phon* verwendet, also als sprachliche Lauteinheit, die als eigenständiges Schallsegment wahrgenommen werden kann.

33 Die Anzahl der Phoneme einer Sprache ist im Extremfall gleich der Zahl der in der Sprache verwendeten Phone. Mitunter lassen sich keine Minimalpaare zur Unterscheidung von Phonen finden (im Deutschen etwa für [ŋ] und [h]). Man spricht in diesem Fall von verschiedenen *Allophonen* desselben Phonems.

34 Zumindest wenn man der *Distinktivitätskonzeption* des Graphembegriffs folgt, wo Grapheme als die kleinsten bedeutungsunterscheidenden Einheiten einer Schriftsprache definiert sind. Die konkurrierende *Repräsentanzkonzeption* dagegen definiert Grapheme als schriftsprachliche Darstellung von Phonemen. Eine Reihe von Gründen, weshalb die Distinktivitätskonzeption der Repräsentanzkonzeption vorzuziehen ist, findet sich bei Günther (1988:76f).

als Merkmale der einzelnen Grapheme gesetzt. Anhand der Verteilung dieser Merkmale können Grapheme durch geeignete Verfahren (z.B. Clusterer oder Klassifikatoren) in Gruppen, die sich durch ähnliche Eigenschaften auszeichnen, zusammengefasst werden.

Ermittlung von Morphemen: Die morphologische Methode

Wie in Kapitel 4.2.3 angeführt, gab es bereits mehrere Überlegungen, die VM-Wörter in einzelne Bestandteile zu zerlegen (Tiltman 1967 und Stolfi 1997b), die wir der Einfachheit halber – leicht sinnentfremdet[35] – Morpheme nennen wollen. Auch die Chiffren der P.III lassen sich als Stammmorpheme interpretieren, an die Derivations- oder Flexionsaffixe angehangen sind (vgl. Anhang A.2). Die Ähnlichkeit in den Wortmodellen legt nahe, einen Versuch zu unternehmen, die Methoden, welche für die Morphemanalyse natürlicher Sprachen entwickelt wurden, auf P.III- und VM-Chiffrentexte anzuwenden. Für die Zerlegung von Wörtern in Morpheme existieren verschiedene Ansätze. Wir nutzen hier die Methodik des Distributionalismus (einer Form des amerikanischen Strukturalismus), die auf dessen zentralen Akteur, den Linguisten Zellig Harris, zurückgeht. Harris (1954:3) definierte das distributionelle Programm wie folgt:

> "Each language can be described in terms of a distributional structure, i.e. in terms of the occurrence of parts (ultimately sounds) relative to other parts, and [...] this description is complete without intrusion of other features such as history or meaning."

Zur Bestimmung des Morpheminventars mit rein distributionellen Mitteln entwickelte Harris (1951)[36] eine Methode zur Ermittlung von Morphemen aus vorhandenen Sprachdaten ohne Rückgriff auf Ressourcen wie Lexika oder Morphemlisten. Eine Umsetzung der Methodik mit angemessenen computationellen Mitteln findet sich bei Benden (2005). Stark vereinfacht lässt sich das Vorgehen wie folgt beschreiben: Sämtliche Wörter werden in einem Baum aggregiert, bei dem die einzelnen Knoten Buchstaben (inkl. Häufigkeiten), die Kanten zwischen den Knoten Übergänge zwischen den Buchstaben repräsentieren. Ein solcher Baum wird auch Keyword-Tree genannt.[37] Wenn alle Wörter aufgetragen sind, kann man für jeden Knoten bestimmen, wie viele Nachfolgeknoten (Successor Counts,

35 Natürlich handelt es sich nicht direkt um Morpheme als kleinste bedeutungstragende Einheiten, vgl. dazu auch Anhang A, Fußnote 1.

36 Die Methode wurde später von ihm selbst (Harris 1954) weiter ausgearbeitet.

37 Im Unterschied zu dem in einer späteren Analyse (6.3) verwendeten SuffixTree werden in KeywordTrees nur ganze Wörter, nicht alle Suffixe aufgetragen.

SC) er hat. Invertiert man die Wörter, bevor man sie in den Baum aufträgt, so erhält man die Anzahl der Vorgängerknoten (Predecessor Counts, PC).

Benden (2006) analysierte mit dieser Methode ein selbst (aus deutschen Zeitungstexten) zusammengestelltes Korpus von 100 Texten. Nach dem Auftragen der Wörter in zwei Keyword-Trees (einer vorwärts zur Ermittlung der SC, einer rückwärts für die Ermittlung der PC) finden sich die Werte aus Tabelle 6.9 auf dem Pfad des Wortes *gegenständlichen*.

	g	e	g	e	n	s	t	ä	n	d	l	i	c	h	e	n
SC	14	22	5	3	8	5	3	1	1	2	1	1	1	1	1	0
PC	0	1	1	1	1	1	1	1	5	6	12	4	15	11	25	16

Tabelle 6.9: *Successor (SC) und Predecessor Counts (PC) für die Zeichenfolge "gegenständlichen", aus Benden (2006:321). Rot eingefärbt die Werte, die auf Morphemgrenzen nach (SC) bzw. vor (PC) dem Element hindeuten.*

Dort, wo die Werte besonders hoch sind bzw. dort, wo sie ansteigen, werden Morphemgrenzen angenommen. Die Idee dahinter ist, dass die Variabilität möglicher Vorgänger oder Nachfolger an Morphemgrenzen zunimmt, weil Morpheme sich intern sehr konstant verhalten, an ihren Grenzen aber relativ frei kombinierbar sind. Für das Beispiel aus Tabelle 6.9 ergäbe sich die Morphemanalyse *ge-gen-ständ-li-ch-en*, wobei die ersten beiden Morphemgrenzen aus den SC-Werten folgen, die letzten beiden aus den PC-Werten, während die Grenze zwischen *ständ* und *lich* von beiden Reihen von Werten unterstützt wird. Die Analyse entspricht nicht ganz der traditionellen Morphemzerlegung, die *ge-gen* und *li-ch* so nicht getrennt hätte. Die Ursachen für diese Fehlanalysen liegen wahrscheinlich darin begründet, dass sowohl *ge-* als auch *-chen* im Deutschen gebräuchliche Affixe (z.B. Partizip-II-Konstruktion und Verniedlichungsform) sind, diese aber nicht in dem analysierten Wort vorkommen. Statt wie in diesem Fall zuviele, werden in anderen Konstellationen zuwenige Morphemgrenzen erkannt. Benden (2006) skizziert deshalb ein mehrstufiges Verfahren, bei dem im ersten Schritt die SC/PC-Werte ermittelt werden, um ein initiales Morpheminventar zu erhalten. Im zweiten Schritt werden die Wörter anhand dieses Inventars analysiert und sämtliche mögliche Morphemzerlegungen gesammelt. In einem dritten Schritt wird die wahrscheinlichste Zerlegung durch einen auf verschiedenen linguistischen Parametern basierenden Selektionsprozess ausgewählt.

Übertragen auf die hier zu analysierenden Chiffrentexte muss sich zeigen, ob diese Methodik auch bei Texten, die fast ausschließlich aus kürzeren Wörtern bestehen, in der Lage ist, Morpheme innerhalb dieser Wörter zu detektieren. Theoretisch bilden VM- und P.III-Text

eine abgeschlossene Sprache, so dass die Erzeugung eines vollständigen Morpheminventars dieser Sprache möglich erscheint. Auch der sehr regelmäßige Aufbau der Wort-Chiffren deutet darauf hin, dass dieses Vorhaben Erfolg haben könnte. Morpheme sind elementare Konstruktionsmerkmale der P.III-Tabellen; ihre Ermittlung wäre möglicherweise auch sehr hilfreich bei der Re-Konstruktion einer potentiellen VM-Tabelle.

6.2.2 Hypothesen

Die übergeordnete Hypothese, die in diesem Kapitel experimentell überprüft werden soll, behauptet, dass die oben aufgeführten strukturalistischen Methoden bei der Abbildung von Chiffren auf P.III-artige Tabellen behilflich sind. Dazu muss zunächst der Mechanismus untersucht werden, bei dem die Tabellenstruktur bekannt ist. Daraus ergeben sich dann im Einzelnen die folgenden Hypothesen:

- Die für die Zuordnung von P.III-Wörtern zu Klartextbuchstaben relevanten Einheiten (Vokale, Schlusskonsonanten) lassen sich durch distributionelle Analyse Minimalpaar-unterscheidender Einheiten (Grapheme) ermitteln.
- Die für die Zuordnung von P.III-Wörtern zu Chiffrensets (=Spalten der Ersetzungstabelle) relevanten Stammmorpheme lassen sich durch distributionelle Analyse von Wortbestandteilen (Morphemen) gewinnen.

Liefern die Verfahren im Fall von P.III-Wörtern akzeptable Ergebnisse, so können sie möglicherweise auch auf VM-Wörter angewendet werden:

- Beide Methoden lassen sich auf VM-Grapheme und VM-Wörter übertragen und liefern damit wichtige Anhaltspunkte für die Rekonstruktion der vermuteten VM-Substitutionstabellen.

6.2.3 Design der Experimente

Für die Überprüfung der Hypothesen bedarf es der Implementation der beschriebenen Verfahren in zwei Workflows, einem für den graphematischen, einem für den morphologischen Ansatz. Beide Verfahren werden an einer P.III-Chiffre evaluiert und anschließend auf den VM-Text (Takahashi-Transkription, EVA- und Currier-Alphabet) angewendet.

Bei der Graphemanalyse werden sämtliche Wortpaare mit minimaler Differenz (Austausch oder Fehlen eines Buchstabens) analysiert: Wo befindet sich das Minimalpaar-unterscheidende Element relativ zum Wortanfang und Wortende, welche Buchstaben bil-

den den Kontext, durch welches Element wird es im anderen Wort ersetzt, in wie vielen Minimalpaaren kommt es vor? Für jedes Graphem werden Vektoren erzeugt, auf welche die einzelnen Merkmale oder Kombinationen von Merkmalen aufgetragen werden. Diese Vektoren können über Cluster-Verfahren gruppiert werden, um zu erreichen, dass Grapheme mit ähnlichen Merkmalen in dasselbe Cluster einsortiert werden.

Bei der Morphemanalyse werden sämtliche Types aus den Texten auf zwei Keyword-Trees aufgetragen, einen für die Ermittlung der SCs, einen für die der PCs. Die Art der Verteilung von PCs und SCs wird ausgewertet, um Morphemgrenzen zu detektieren und damit ein initiales Morphemset zu erzeugen. Bei dieser Auswertung können verschiedene Charakteristika als Hinweise auf Morphemgrenzen eine Rolle spielen: Wo befinden sich lokale oder absolute Maxima in der Verteilung, wo steigen Werte an, wo fallen sie ab? Jede ausgewählte Charakteristik, die auf eine Morphemgrenze hindeutet, erhöht den *Score* einer bestimmten Position. Erreicht der Score einen Schwellwert, wird eine Morphemgrenze angenommen. Mit dem initialen Morphemset werden in einem zweiten Schritt sämtliche möglichen Morphemzerlegungen der zu analysierenden Wörter erzeugt. In einem dritten Schritt werden – anhand der Häufigkeit des Auftretens von Morphemen in Wörtern – Kandidaten für Stammmorpheme ausgemacht.

Abbildung 6.3 zeigt ein Experiment, in dem beide beschriebenen Verfahren umgesetzt sind. Die Gruppierung der vom Graphemizer erzeugten Grapheme wird von einem Clusterer übernommen, während die Analyse der vom Morphemizer produzierten Morpheme über eine manuelle Sichtung der Ausgabe erfolgt. Eine maschinelle Analyse der Morpheme wurde zwar bereits angedacht,[38] für die Umsetzung im Rahmen dieser Arbeit aber noch nicht genutzt.

6.2.4 Benötigte Komponenten

Wie in 6.1 werden **Reader**, **Präprozessor** und **Verschlüsselungskomponente** genutzt, um die zu analysierenden Texte einzulesen, zu tokenisieren und gegebenenfalls mit der P.III-Methode zu verschlüsseln. Die beschriebenen distributionellen strukturalistischen Verfahren wurden jeweils in eigenen Komponenten gekapselt, dem **Graphemizer**

38 Dazu wurden Morpheme durch einzelne Zeichen repräsentiert, die dann Eingabe für die distributionelle Analyse im Graphemizer wurden, dessen Output geclustert werden kann – die Komponentenabfolge dieses Experiments wäre Reader-Präprozessor-Morphemizer-Graphemizer-Clusterer. Ebenso wurde die technische Möglichkeit geschaffen, die Komponenten in umgekehrter Richtung zu verschalten, indem nämlich die Cluster von Graphemen in die Berechnung der Morphemgrenzen einbezogen wurden. Näheres dazu findet sich in der Komponentenbeschreibung des *Extended Morphemizer*, Anhang C.

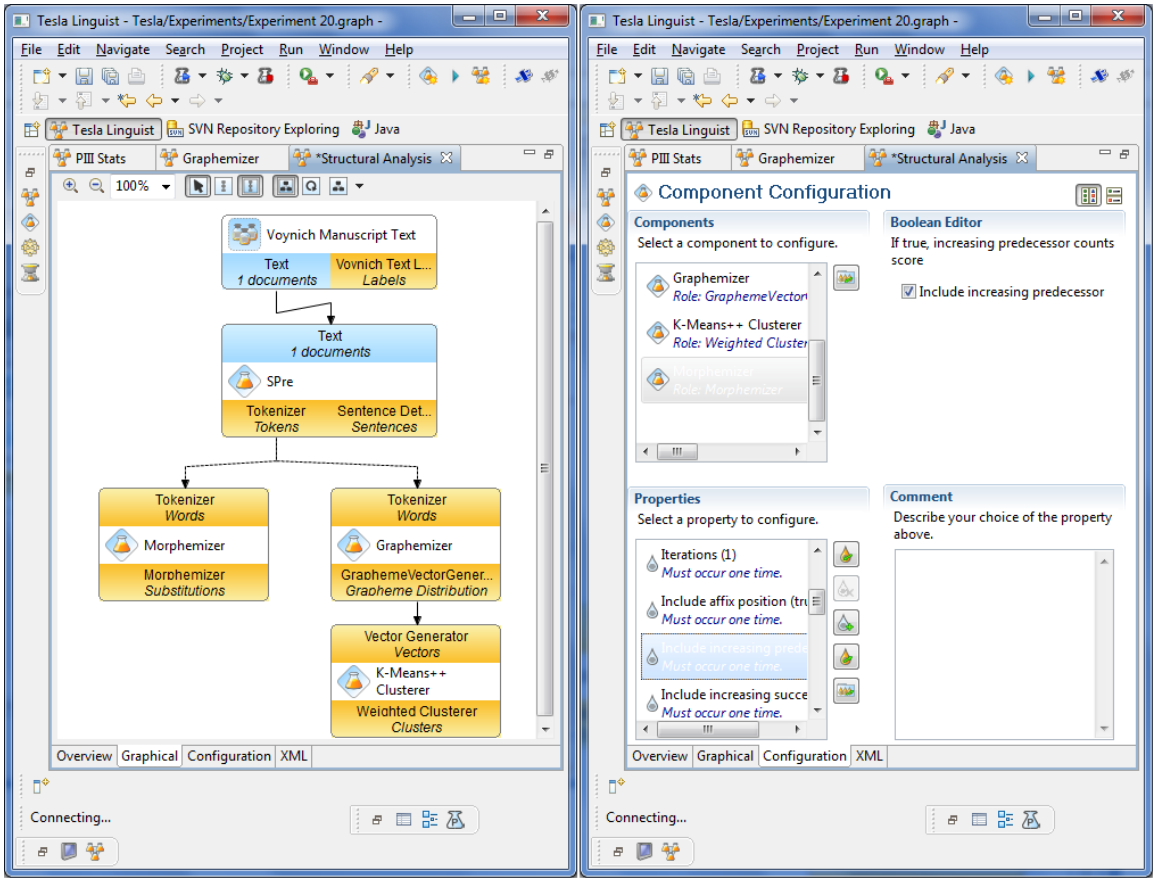


Abbildung 6.3: *Einbettung der beiden im Text beschriebenen strukturalistischen Aufdeckungsverfahren zur Ermittlung von Graphem- und Morpheminventar, zusammengefasst in einem Experiment. Auf der linken Seite der graphische View, auf der rechten Seite ein Blick auf die Konfigurationsmöglichkeiten des Morphemizers.*

und dem **Morphemizer**. Für das Clustering der vom Graphemizer erzeugten und mit Merkmalen versehenen Grapheme wird darüber hinaus eine **Clustering-Komponente** benötigt.

Graphemizer

Der Graphemizer identifiziert sämtliche Minimalpaare des zu bearbeitenden Textes und leitet daraus ein Set von Graphemen (als Minimalpaar-unterscheidende Einheiten) ab. Technisch gesehen werden dafür alle Paare von Types auf ihre Levenshtein-Distanz[39] ge-

39 Die Levenshtein-Distanz ist ein Maß für die Ähnlichkeit von zwei Zeichenketten. Sie drückt aus, wie groß die minimale Anzahl von Einfüge-, Lösch- und Ersetz-Operationen ist, um die erste Zeichenkette in die zweite umzuwandeln.

testet. Paare mit der Levenshtein-Distanz *1* werden auf ihre unterscheidende Einheit hin untersucht. Aus der Distribution der Grapheme in ihren Minimalpaar-unterscheidenden Kontexten werden weitere Merkmale abgeleitet, die über die Konfiguration bestimmt werden: Als Merkmale gewählt werden können die Minimalpaar-Partner (s.o.), die Glyphen des direkten Kontextes (welcher Glyph steht vor, welcher nach dem Graphem), die Position vom Wortanfang und die vom Wortende aus gezählt. Außerdem kann die Zahl der Minimalpaare aufgenommen werden, in denen das Graphem unterscheidend ist. Über eine weitere Konfigurationsschnittstelle können außerdem kurze Wörter aus der Analyse ausgeschlossen werden.

Die Komponente erzeugt für jedes ermittelte Graphem ein Datenbankobjekt, das sowohl eine symbolische Darstellung als auch eine numerische Repräsentation seiner Merkmale enthält. Erstere ist hilfreich bei der manuellen Auswertung von Graphemen, letztere erlaubt den Einsatz von Clusterern oder Klassifikatoren, die auf numerischen Daten operieren.

Clusterer

In Tesla wurden bereits eine ganze Reihe verschiedener Clusterer implementiert, unter anderem wurde die WEKA-API[40], die eine Reihe von unterschiedlichen Cluster- und Klassifikationsverfahren zur Verfügung stellt, eingebunden. Für das Clustering der Grapheme ist der Einsatz der `KMeansClusteringComponent` ausreichend, welche den `KMeansPlusPlusClusterer` des apache.commons.math-Projektes nutzt und nur ein Verfahren (KMeans++) anbietet, das aber sehr effizient arbeitet und eine Reihe von Konfigurationsmöglichkeiten bietet. Bei der Komponente sind Abstandsmaß sowie Anzahl der Cluster und Iterationen frei wählbar, was einen flexiblen Einsatz der Komponente ermöglicht.

Morphemizer

Der Morphemizer identifiziert über die distributionelle Methdode ein initiales Set von Morphemen zu den Types des gegebenen Textes. Konkret werden dabei zwei Keyword-Trees aus sämtlichen Types des gegebenen Textes gefüllt (einer der Trees wird mit den Types selbst, der andere mit den inversen Types gefüllt). Aus diesen ergeben sich für jeden Buchstaben jedes Types spezifische Successor- und Predecessor-Werte, die für die Zerlegung in Morpheme genutzt werden. Über die Konfiguration kann gesteuert werden,

40 Siehe `http://sourceforge.net/projects/weka/`, zuletzt aufgerufen am 11.10.2011.

welche Merkmale auf eine Morphemgrenze hinweisen (Score) und wie viele dieser Merkmale erfüllt sein müssen, um tatsächlich eine Morphemgrenze anzunehmen. Successor- und Predecessor-Werte können einzeln gewählt werden, zusätzlich kann man beide Werte positionsgenau addieren und auch diese Summe auswerten. Dazu kann man nur ansteigende bzw. abfallende Positionswerte als Merkmal annehmen, aber auch die Positionen, an denen sich absolute oder lokale Maxima befinden. Schließlich kann man noch festlegen, ob jeder Type in höchstens zwei Morpheme geteilt werden und ob das Verfahren iterativ arbeiten soll (dabei werden sämtliche ermittelten Morpheme in die beiden Trees eingetragen und die Analyse beginnt von neuem).

Die Komponente arbeitet für unseren Anwendungsfall dreischrittig: Zunächst wird über die Auswertung der distributionellen Daten ein initiales Set von Morphemen generiert. Mit diesem Morphemset werden die Wörter des Textes in alle möglichen (d.h. vom Morphemset gedeckten) Morphemabfolgen zerlegt. Im dritten Schritt wird die Häufigkeit von Morphemen in den möglichen Zerlegungen bestimmt.

6.2.5 Ergebnisse

Die Ergebnisse der graphemischen und morphologischen Analyse einer P.III-Chiffre und des VM-Textes finden sich in den anschließenden Abschnitten. Exemplarisch wurden die Methoden auch auf einen natürlichsprachlichen Text angewendet, um auch hier noch einmal herauszustellen, wie sehr beide Chiffren sich in ihrem Unterschied zu natürlichsprachlichen Texten gleichen. Abschließend werden die gewonnenen Erkenntnisse diskutiert.

Clustering der P.III-Grapheme

Das Ziel bei der Anwendung der Graphemanalyse auf die P.III-Chiffre war die Ermittlung von drei P.III-Graphem-Clustern durch ein rein distributionelles Verfahren. Idealerweise bekäme man ein Cluster mit den Vokalen, ein weiteres Cluster mit den Konsonanten, die als Schlusskonsonanten verwendet werden und schließlich ein Cluster für die restlichen Konsonanten. Wie oben beschrieben, wurden die Grapheme über eine Minimalpaaranalyse ermittelt. Die Kontexte, in denen sie als Minimalpaar-unterscheidende Einheiten (MPuE) gefunden wurden, werden als Merkmale interpretiert. Dabei wurden drei verschiedene distributionelle Eigenschaften ermittelt:

- An welcher *Position* im Wort (sowohl von vorne wie auch von hinten gezählt) befindet sich die MPuE?

- Welches ist die MPuE im komplementären Wort des Minimalpaars (*Partner*)?
- In welchem *Kontext* (Zeichen davor, Zeichen dahinter) befindet sich die MPuE?

Als Clusterverfahren wurde die K-Means++-Methode (3 Cluster, euklidische Distanz, 50 Iterationen) aus der gleichnamigen Tesla-Komponente verwendet. Die Ergebnisse finden sich in Tabelle 6.10. P.III-Chiffren sind derart regelmäßig aufgebaut, dass sich die Cluster, die auf den verschiedenen Merkmalen basieren, nicht von den Clustern, die auf der Kombination aller Merkmale beruhen, unterscheiden. Auch das Ziel, Vokale und Schlusskonsonanten in divergente Cluster zu unterteilen, ist zum großen Teil gelungen, die Vokale finden sich alle jeweils im ersten, fast sämtliche Schlusskonsonanten im zweiten Cluster.

	Cluster 1	Cluster 2	Cluster 3
Positionen	a e i o u	l n r s t x y $	b c d f g h m p v z
Partner	a e i o u	l n r s t x y $	b c d f g h m p v z
Kontexte	a e i o u	l n r s t x y $	b c d f g h m p v z
Alle	a e i o u	l n r s t x y $	b c d f g h m p v z

Tabelle 6.10: *Ergebnisse der Graphemanalyse auf einer P.III-Chiffre. Gut zu sehen ist, dass die Analyse sämtlicher Merkmale zu den gleichen Clustern führt. $ bezeichnet eine Lücke (fehlender Buchstabe beim Minimalpaar-Vergleich).*

Die beiden einzigen Falschzuordnungen sind f und m im dritten Cluster. Sie werden, wenn auch sehr spärlich,[41] in der P.III als Schlusskonsonanten genutzt. Während die Analyse das f nicht richtig erfasst, weil es von Trithemius zum Wortausklang immer verdoppelt wird (Bsp. *loraff*) und damit kein Minimalpaar bilden kann,[42] wird das m schlichtweg zu selten als Schlusskonsonant eingesetzt, während es öfter zur Unterscheidung von Stämmen genutzt wird. Es ist den übrigen stammunterscheidenden Konsonanten (wie z.B. dem b) deshalb ähnlicher als den anderen Schlusskonsonanten (wie z.B. dem x). Der Unterschied lässt sich übersichtlich anhand der Positionsdistributionen (Tabelle 6.11) darstellen.

Lässt man statt drei fünf Cluster berechnen, so bleiben die ersten beiden Cluster konstant, lediglich z und g bilden eigene Cluster. Die beiden Grapheme sind diejenigen, welche die wenigsten Minimalpaare – und zwar ausschließlich auf der ersten Position – unterscheiden. Von der Erhöhung der Clusterzahl ist demnach keine für unseren Kontext signifikante Information zu erwarten.

41 Das f wird in zwei Blöcken, das m gar nur in einem Block genutzt.

42 Womit sich ein Nachteil des Verfahrens zeigt, das Grapheme immer nur als einzelne Zeichen annimmt, anders als etwa Günther (1988) für das Deutsche, s.o.

	n	Positionsdistribution (von vorne)	Positionsdistribution (von hinten)
a	2557	0.02-0.08-0.04-0.41-0.19-0.21-0.04	0.15-**0.74**-0.02-0.06-0.02-0.01-0.00
x	516	0.00-0.00-0.04-0.00-0.58-0.10-0.28	**1.00**-0.00-0.00-0.00-0.00-0.00-0.00
b	364	**0.61**-0.07-0.32-0.00-0.00-0.00-0.00	0.00-0.04-0.26-0.12-0.57-0.00-0.02
m	76	**0.49**-0.00-0.19-0.07-0.13-0.08-0.04	0.29-0.03-0.19-0.07-0.41-0.00-0.00

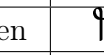
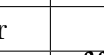
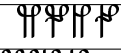
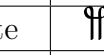
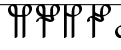
Tabelle 6.11: *Distribution der Position ausgewähler Grapheme mit Anzahl (n) der Minimalpaare, in denen das Graphem die unterscheidende Einheit ist, sowie den Positionen, in denen es als MPuE gefunden wurde (relative Werte), einmal vom Wortanfang, einmal vom Wortende aus gezählt.* 'a' *ist ein typischer Vokal, der an letzter oder vorletzter Stelle die MPuE bildet,* 'x' *ein typischer Schlusskonsonant, der MPuE an der letzten Stelle ist.* 'b' *wird nicht als Schlusskonsonant genutzt, unterscheidet aber Stämme, v.a. auf den Positionen 1 und 3.* 'm' *wird für beide Zwecke eingesetzt, ist* 'b' *aber ähnlicher.*

Clustering der VM-Grapheme

Um die Problematik der Zuordnung von VM-Glyphen zu Graphemen (vgl. 4.2.2) zu berücksichtigen, bietet sich an, die Graphem-Distributionsanalyse auf den VM-Text in verschiedenen Transkriptionsalphabeten anzuwenden. Darüber hinaus erscheint es zweckmäßig, den Text für beide Currier-Sprachen (vgl. 4.2.3) getrennt zu analysieren. Die Clusteranalysen von D'Imperio (1978b) deuteten darauf hin, dass sich die Teile grundlegend unterscheiden, so dass ihnen möglicherweise verschiedene Verschlüsselungsmethoden (in unserem Fall verschiedene Ersetzungstabellen) zugrunde liegen. Sollte das so sein, dann würde eine gemeinsame Behandlung beider Teile in derselben Analyse die Ergebnisse verwässern. Die Merkmale werden wieder zunächst einzeln, danach gemeinsam betrachtet. Schon bei drei Clustern tendieren die Grapheme ı und ϱ dazu, Cluster alleine zu belegen. Deshalb wurden die Grapheme insgesamt auf fünf statt auf drei Cluster verteilt. Tabelle 6.12 zeigt die ermittelten Graphemcluster für das EVA-Alphabet.

Das Ergebnis ist nicht so eindeutig wie das für den P.III-Text, ohne Zweifel sind aber gewisse Tendenzen zu erkennen: Die Gallows (zur Erinnerung: die stark vertikal ausgeprägten Zeichen) finden sich bei jeder einzelnen Merkmalsanalyse in einem gemeinsamen Cluster, lediglich die Kombination der Merkmale im Currier-B-Teil reisst diese Gruppe auseinander.[43] In allen Fällen findet sich auch das ϲ in diesem "Gallows-Cluster", vereinzelt auch das τ. Beide zusammen bilden das stark vertikal ausgerichtete Zeichen (ϲτ) das

43 Hier zeigt sich eine bemerkenswerte Eigenschaft des Clusterverfahrens, dass es nämlich bisweilen unvorhergesehene Ergebnisse liefert. Bei allen einzelnen Merkmalen findet man die Gallows im gleichen Cluster. Bei Kombination genau dieser Merkmale finden sie sich plötzlich in zwei verschiedenen Clustern.

Currier A	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Positionen					
Partner					
Kontexte					
Alle					

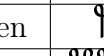
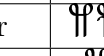
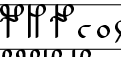
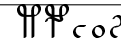
Currier B	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Positionen					
Partner					
Kontexte					
Alle					

Tabelle 6.12: *Ergebnisse der Graphemanalyse auf dem VM, EVA-Alphabet. Analysen wurden für die Currier-A- und -B-Teile des Textes getrennt durchgeführt.*

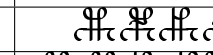
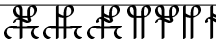
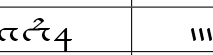
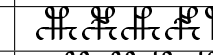
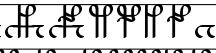
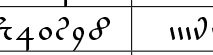
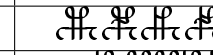
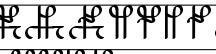
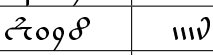
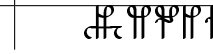
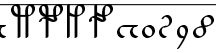
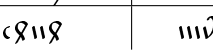
oft Ligaturen mit den Gallows bildet. Das ı-Zeichen bildet entweder allein ein Cluster oder in Verbindung mit den anderen "Vokalen" a und c, teilweise gesellen sich noch τ und ℴ dazu. Zeichen, die mit einem oder mehreren ı kombiniert werden können (ϑʃʁʔ) finden sich meist in zwei Cluster aufgeteilt. Schon im sehr analytischen gehaltenen EVA-Alphabet, das der Graphemanalyse nicht gerade entgegenkommt, sind also gewisse Regelmäßigkeiten zu entdecken. Die Ergebnisse der gleichen Versuchsanordnung mit einer abweichenden Reader-Konfiguration - der VM-Text wird im Currier- statt im EVA-Alphabet geliefert findet sich in Tabelle 6.13.[44]

Aus den Clusterings der Currier-Alphabet-Grapheme resultiert fast durchgehend eine Trennung von Gallows und Gallow-Kombinationen von den Zeichen, die mit einem oder mehreren ı kombiniert werden können. Currier A bildet außerdem noch ein drittes Cluster aus, in dem sich die EVA-"Vokale" mehr oder weniger vollständig wiederfinden (bei Currier B finden sich stattdessen im dritten Cluster weitere ı-Kombiantionen).

Clustering natürlichsprachlicher Grapheme

Zu Vergleichszwecken wird die oben beschriebene Prozedur auf natürlichsprachliche Texte angewendet, und zwar auf verschiedene Bibeltexte (lateinisch) sowie Friedrich Nietzsches *Ecce Homo* (deutsch). Als auffällige Diskrepanz zu den vorher untersuchten Chiffren fällt

44 Zur Erinnerung: Currier vergab für jede Ligatur ein eigenes Graphem. Die Zahl der Grapheme ist also ungleich höher.

Currier A	Cluster 1	Cluster 2	Cluster 3
Positionen			
Partner			
Kontexte			
Alle			

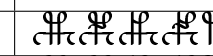
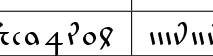
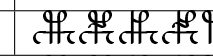
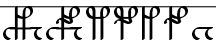
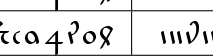
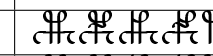
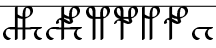
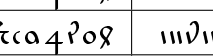
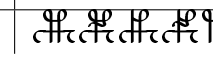
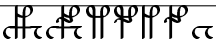
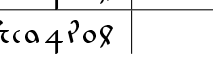
Currier B	Cluster 1	Cluster 2	Cluster 3
Positionen			
Partner			
Kontexte			
Alle			

Tabelle 6.13: *Ergebnisse der Graphemanalyse auf dem VM, Currier-Alphabet. Analysen wurden für die Currier-A- und -B-Teile des Textes getrennt durchgeführt. Insgesamt wurden fünf Cluster erstellt, die letzten beiden sind aus Platzgründen nicht dargestellt. Sie enthalten lediglich die sehr raren Grapheme*  *und*

direkt ins Auge, dass die Anzahl von Minimalpaaren bei natürlichen Sprachen sehr viel geringer ist:[45] Der gewählte P.III-Text besteht aus etwa 11200 Tokens (ca. 2000 Types) und weist insgesamt über 8000 Minimalpaare auf. Der VM-Text, Currier-A-Teil besteht aus etwa der gleichen Anzahl (11400) Tokens, weist 3446 Types und mehr als 9500 Minimalpaare auf. Obschon der Text von Nietzsche weit mehr Tokens (54000) und Types (6000) aufweist, finden sich sehr viel weniger Minimalpaare (3170), ebenso beim lateinischen Bibel-Teil *Exodus* (20000 Tokens, 4700 Types, 2300 Minimalpaare).

Die Ergebnisse des Graphem-Clusterings sind nur eingeschränkt interpretierbar und werden aus diesem Grund vergleichsweise kompakt dargelegt. Für Latein ergab die Abbildung auf fünf Cluster bei Berücksichtigung sämtlicher Merkmale die Graphemverteilung *(q)-(x)-(r c b g)-(d v l f h p)-(o t i e n u s m a $)*. Für den deutschen Nietzsche-Text wurden die folgenden Cluster ermittelt: *(y)-(l d v p b w j h f k g z)-(t m r s n e $)-(ö ä ü a i o x c)*. Im deutschen Text werden also tatsächlich die Vokale (inklusive Umlaute, aber leider ohne das *e*) in einer Gruppe zusammengefasst. Dennoch wird an den Ergebnissen deutlich, dass das Verfahren für die gewählten natürlichsprachlichen Texte keine so gut interpretierbaren Ergebnisse produziert, wie dies durch die Anwendung auf die oben untersuchten Chiffrentexte gelungen ist. Möglicherweise können die Ergebnisse verbessert werden (so dass z.B. ein Cluster sämtliche Vokale und keine weiteren Konsonanten enthält), indem

45 Diese Erkenntnis wurde in 6.1 so noch nicht thematisiert, dort ging es nur um *adjazente* Minimalpaare.

längere Texte untersucht werden und damit die Zahl der Minimalpaare steigt.[46]

Morphologische Zerlegung der P.III-Chiffren

Die morphologische Zerlegung sollte am besten funktionieren, wenn alle gültigen Wörter einer Sprache in die Keyword-Trees aufgetragen werden. Bei natürlichen Sprachen ist dies nicht möglich, da die Zahl der Wörter im Prinzip unendlich groß ist. Im Fall der P.III-Methode kann dies dennoch gelingen, wenn alle Chiffren, die in den P.III-Tabellen auftreten, im zu analysierenden Text vorkommen. Im Normalfall wird dies so nicht eintreten (es sei denn der Text ist sehr lang oder das Auswahlverfahren ist entsprechend angepasst), für unsere erste Analyse wollen wir dennoch zunächst einen solchen Text nutzen, um die Tauglichkeit der Morphemanalyse auf P.III-Chiffren ausloten zu können.

Tabelle 6.14 zeigt Werte für einige ausgewählte exemplarische P.III-Chiffren, die aus Keyword-Trees gewonnen wurden, welche mit sämtlichen P.III-Types gefüllt wurden. Die Zerlegungen zeigen, dass das Verfahren (Scoring sämtlicher Merkmale, Score-Wert >= 3 bewirkt Zerlegung) schon auf der ersten Stufe relativ gute Ergebnisse generiert. *Pasan* und *pasa* werden exakt in das gewünschte Morphemmodell (Stamm - letzter Vokal - fakultativer Schlusskonsonat) zerlegt, bei *husal* wird zumindest der Stamm korrekt erkannt. Die Zerlegung *nas* zeigt, dass auch sehr kurze (nur aus einem Zeichen bestehende) Stämme erkannt werden können, während man bei der von *cabalas* sieht, dass lange Stämme dazu tendieren, in zwei Einheiten getrennt zu werden.

Chiffre	**Successor-Werte.**	**Predecessor-Werte**	**Score**	**Zerlegung**
pasan	6 3 5 6	3 7 11 5	1 1 5 3	pas-a-n
pasa	6 3 5	3 7 12	1 1 4	pas-a
husal	4 2 5 5	1 7 10 5	0 1 7 2	hus-al
nas	6 5	11 5	5 0	n-as
cabalas	3 3 5 5 6 5	1 1 3 2 11 5	0 0 4 0 7 0	cab-al-as

Tabelle 6.14: *Successor- (SV) und Predecessor- (PV) Werte ausgewählter P.III-Chiffren. SVs sind von vorne nach hinten, PVs von hinten nach vorne zu lesen. Als Score wurden sämtliche Merkmale berücksichtigt (siehe oben).*

Die einzelnen ermittelten Morpheme werden nach Präfixen, Infixen und Suffixen getrennt

46 Gegebenenfalls müsste die Graphemizer-Komponente für die Untersuchung längerer Texte grundlegend überarbeitet werden. Momentan wird für jedes Type-Paar die Levenshtein-Distanz errechnet, was zu einer quadratischen Laufzeit (bezogen auf die Type-Anzahl) führt.

in Listen aggregiert. Zu jedem Morphem wird die Anzahl der Vorkommen in verschiedenen Wörtern gezählt.[47] Die Zahl ist entscheidend für den zweiten Schritt, wo das initiale Morphemset auf jedes Wort des zu analysierenden Textes angewendet wird, um mögliche Morphemzerlegungen zu ermitteln (siehe Tabelle 6.15).

Chiffre	**Stamm + Suffix**	**Stamm + 2 Suffixe**	**Falscher Stamm**
pasan	pas-an [25-82]	pas-a-n [25-96-56]	pasa-n [1-56]
pasa	pas-a [25-96]		pa-sa [0-5]
husal	hus-al [24-83]	hus-a-l [24-96-56]	husa-l [0-56]
nas	n-as [24-89]	n-a-s [24-96-56]	na-s [0-56]
cabalas	cabal-as [4-89]	cabal-a-s[4-96-56]	cab-al-as [44-35-89]

Tabelle 6.15: *Mögliche Morphemzerlegungen ausgewählter P.III-Chiffren. Die Werte in Klammern geben an, wie oft das Morphem im initialen Set als Präfix, Infix oder Suffix gefunden wurde. Sie können als Grundlage eines Auswahlprozesses dienen.*

Schließt man Zerlegungen mit sehr niedrigen Werten aus der Auswahl aus, so erkennt die Methode die P.III-Stämme relativ zuverlässig: Der falsche Stamm (*pasa-*) etwa kommt nur einmal vor, während der richtige (*pas-*) in 25 Fällen gefunden wird. Für *husal* und *nas* wird gar kein falscher Stamm gefunden. Auch auf dieser zweiten Stufe der Analyse tendieren längere Stämme (hier *cabalas*) wieder dazu, zwei Einheiten zu bilden. Der Grund liegt darin, dass *cab-* weit häufiger vorkommt als *cabal-* und deswegen fälschlicherweise als Stamm angenommen wird.

In einem dritten Schritt aggregiert man die Morpheme aller (alternativ: nur die der besten) möglichen Zerlegungen. Damit erhält man die Anzahl der Wörter, in denen das Morphem potentiell vorkommt. Im Fall des analysierten P.III-Textes, der alle Chiffren enthält, kommen manche Morpheme nur in einem einzigen Wort vor, andere sehr häufig. An der Spitze der Häufigkeitsverteilung steht das *a*, das in 796 von 2875 Wörtern als mögliches Morphem vorgeschlagen wird. Interessant aber ist v.a. die größte Gruppe mit Morphemen gleicher Häufigkeit: Ganze 70 Morpheme sind möglicher Bestandteil von genau 24 Wörtern. Mit dem Wissen, dass Trithemius 24 verschiedene Buchstaben mit der Chiffrenreihe einer Spalte mit demselben Stammmorphem verschlüsselte, kann man

47 Beispielsweise kommt etwa das Morphem *pas* in zwei Chiffren der Tabelle 6.14 vor, es würde also zweimal gezählt. Dazu könnte man die Höhe des Score (etwa durch Multiplikation mit der Häufigkeit) mit einbeziehen, was im vorliegenden Beispiel aus Übersichtsgründen nicht gemacht wurde. Aus dem gleichen Grund wurde kein Gebrauch von den überaus zahlreichen Konfigurationsmöglichkeiten der Komponente gemacht, etwa die Modifikation des Scoring, die iterative Anwendung oder die Möglichkeit, Morphemgrenzen nur am Score-Maximum anzusetzen.

darauf schließen, dass mit dieser Methode 70 Stammmorpheme gefunden wurden. Diese sind im Einzelnen:

> abr, adram, af, alap, all, amen, asch, astrof, atrop, bad, bai, bar, bed, bem, bod, ch, cob, com, dob, dol, elam, eloh, fas, fesi, flagar, fodr, gened, ger, guar, halmod, harm, hasach, heli, helm, hor, hos, hub, hus, iob, lar, lemor, lod, lusan, marach, masar, mastr, mod, mos, nomys, nys, ophir, pad, pan, perod, pes, plan, poder, pros, rass, rham, rod, sebar, sm, solam, thosar, van, var, vasm, zab, zan

Durch dieses simple Zählverfahren sind also bereits mehr als die Hälfte der von Trithemius eingesetzten Stämme identifiziert.[48] Bei der Analyse der Wörter dürfte der Kryptoanalytiker auch keine Probleme mehr damit haben, die Funktionsweise der Chiffre aufzudecken.

Der Versuch, das Ergebnis mit einem real verschlüsselten Text, der nicht alle Chiffren enthält, zu reproduzieren, führt naturgemäß zu einem etwas schlechteren Ergebnis, weil nun kein Stamm mehr potentieller Teil von genau 24 Wörtern ist. Allerdings ist das Ergebnis nicht viel schlechter, denn noch immer finden sich die meisten Stämme in der gleichen Häufigkeitsklasse, nur etwas weiter gestreut. Aufgelistet sind im Folgenden alle Morpheme, die zwischen 16 und 21 mal vorkommen:

> basch, fesi, gened, iob, pan, amen, astrof, dob, flagar, halmod, hos, lod, nes, nomys, perod, var, adram, alap, asch, atrop, bar, bem, bod, ch, elam, ger, harm, hor, hub, hus, loras, lusan, pad, pes, plan, poder, rod, sebar, solam, zab, bad, bed, cadal, com, eloh, faner, fodr, gom, guar, hasach, heli, masar, mastr, mos, nedri, nys, raf, rass, thosar, van, abr, bai, cabal, dol, eb, helm, ix, lar, lemor, mod, nem, omen, pros, rham, zan

Das Ergebnis ist – durch die minimal weitere Streuung – ein wenig schwerer interpretierbar. Dennoch lässt sich behaupten, dass die strukturalistische Morphemanalyse genau wie die Graphemanalyse bei der Rekonstruktion von P.III-Tabellen wertvolle Dienste leisten kann.

48 Die weiteren Stämme finden sich z.B. in der Häufigkeitsklasse 48 – d.h. sie erstrecken sich über zwei Substitutionsspalten – oder sie wurden nicht in jedem Wort als Morphem erkannt und treten deswegen in weniger als 24 Wörtern auf.

Morphologische Zerlegung der VM-Wörter

Das gleiche Verfahren, das oben auf die P.III-Chiffre angewendet wurde, soll uns nun helfen, mögliche Morpheme der Wörter des VM-Textes offen zu legen. In Tesla gestaltet sich dies recht komfortabel, da lediglich der zu analysierende Text ausgetauscht und der Tokenizer auf das Wortmodell des Voynich-IAF angepasst werden muss. Die Konfiguration des Morphemizers soll exakt die gleiche bleiben, abgesehen davon, dass nicht alle Chiffren, sondern nur die mit mindestens drei Vorkommen in die Analyse mit einbezogen werden (zum Ausschluss möglicher Schreib- bzw. Interpretationsfehler und Füllwörter, s.o.). Tabelle 6.16 zeigt die Successor- und Predecessor-Werte sowie den Score für die VM-Wörter aus den Currier-B-Teilen.[49]

Chiffre	Successor-Werte.	Predecessor-Werte	Score	Zerlegung
cthar	6 1 5 1	1 3 12 6	1 0 6 0	cth-ar
shaiin	7 10 4 2 1	3 11 3 3 2	0 7 0 2 0	sh-aiin
ydaiin	8 2 1 1 1	7 11 3 3 2	2 3 0 2 1	yd-aiin
qokshdy	2 14 9 1 3 2	1 2 4 5 8 11	0 4 0 0 3 1	qo-ksh-dy
qoteedy	2 14 6 4 5 3	2 4 8 6 8 11	0 4 2 0 3 1	qo-tee-dy
qokeedy	2 14 9 6 6 3	4 5 8 6 8 11	0 4 2 0 0 1	qo-keedy

Tabelle 6.16: *Successor- (SV) und Predecessor- (PV) Werte ausgewählter VM-Chiffren. SVs sind von vorne nach hinten, PVs von hinten nach vorne zu lesen. Als Score wurden sämtliche Merkmale berücksichtigt (siehe oben).*

Teilweise ergeben sich recht eindeutige Morphemgrenzen (*cht-ar* und *shd-aiin*), in anderen Fällen sind die Score-Werte gleichmäßiger verteilt (*yd-aiin*). Das Präfix *qo-* wird relativ sicher als Morphem erkannt (Zeilen 3-6 der Tabelle), dafür wird das Suffix *-dy* in zwei der ausgewählten Fälle als solches erkannt, in einem (*qo-keedy*) nicht. Wie man an der Tabelle ablesen kann, liegt das einzig an der Erhöhung eines Successor-Wertes (bei *quokeedy* auf Position 4), so dass sich auf Position 5 kein lokales Maximum befindet, welches den Score inkrementieren würde.

Die Werte zeigen einerseits, dass die Methode auch bei VM-Chiffren Erfolg haben kann, dass auf andererseits aber möglicherweise noch einmal die Konfiguration des Verfahrens überprüft werden müsste, um die Ergebnisse zu verbessern. Mit dieser Erkenntnis korre-

49 Entscheidend sind hier nicht – wie bei der Graphemanalyse – die Zeichen selbst, sondern ihre Kombination. Deshalb werden – im Gegensatz zur Darstellung der Graphemanalyse – die VM-Zeichen hier in ihrer EVA-Transkription aufgeführt, was die Darstellung übersichtlicher macht.

spondiert auch die Tatsache, dass – im Gegensatz zur P.III-Analyse – nur wenige Morpheme in 16-21 verschiedenen Wörtern vorkommen. Stattdessen kommen die meisten Morpheme in weniger Wörtern vor. Dennoch lassen sich aus dieser ersten Analyse schon Schlüsse ziehen, wie die VM-Wörter generell aufgebaut sein könnten. Betrachten wir z.B. exemplarisch das Morphem *ksh* (). In den Currier-B-Teilen des VM findet sich die Zeichenfolge in insgesamt 12 Kombinationen, die mehr als drei Vorkommen haben. Diese Zahl erhöht sich auf 22, wenn auch Types in die Analyse einbezogen werden, die zweifach im Text vorkommen. Diese 22 Kombinationen sind im folgenden aufgelistet:

> d-ksh-ey, ksh-ar, ksh-d, ksh-dy, ksh-ed, ksh-edy, ksh-eeol, ksh-eey, ksh-eody, ksh-ey, l-ksh-edy, l-ksh-ey, o-ksh-edy, o-ksh-ey, o-ksh-y, qo-ksh-d, qo-ksh-dy, qo-ksh-edy, qo-ksh-ey, qo-ksh-y, she-ksh-ey, y-ksh-eol

Die Zahl von 22 verschiedenen Chiffren deutet darauf hin, dass *ksh* einer Art P.III-Stammmorphem entspricht. Aus der Aufzählung von Wörtern lässt sich ersehen, dass es in allen Wörtern mit einem Suffix, dazu in einigen mit einem Präfix kombiniert wird. Anschließende Arbeiten zur Tabellenrekonstruktion könnten anhand von Morphemen mit ähnlichem Verhalten erste Hypothesen über eventuelle Systematiken der Tabellen aufstellen.

Morphologische Zerlegung natürlichsprachlicher Wörter

Die distributionelle Morphemanalyse mit wenigen Texten führt bei natürlichsprachlichen Daten zu sehr dürftigen Ergebnissen, weshalb für die Anwendung des Verfahrens eine breitere Korpusselektion ausgewählt wurde, in die eine Reihe von Texten aus dem Projekt Gutenberg aufgenommen wurden.[50] Weil der Morphemizer (im Gegensatz zum Graphemizer, s.o.) eine lineare Laufzeit hat, kann er in akzeptabler Zeit eine sehr viel größere Textmenge verarbeiten. Insgesamt besteht das Korpus aus mehr als 370.000 Tokens, die sich auf etwa 31500 Types verteilen. Es wurden wieder nur Types in die Analyse mit einbezogen, die mehr als dreimal im Korpus vorkamen, so dass etwa 10.000 Types übrigblieben.

Zunächst wurde die Komponente mit der gleichen Konfiguration auf die deutschen Texte angewendet, die beim VM-Text und vor allem bei der P.III-Chiffre sehr akzeptable

50 Der Auswahl liegt keine tiefere Systematik zugrunde, der Einfachheit halber wurde auf die deutschsprachigen Texte aus dem Gutenberg-Korpus, das bei Tesla als vordefiniertes Korpus mitgeliefert wird, zurückgegriffen. Zu den Autoren gehören Kleist, Nietzsche, Kafka und andere.

Ergebnisse erzeugt hatte. Beim Betrachten der Resultate wird relativ schnell klar, dass natürlichsprachliche Texte eine abweichende Vorgehensweise erfordern – in VM und P.III sind sich die Wörter sehr ähnlich, sowohl hinsichtlich ihrer Länge als auch in der möglichen Abfolge von Graphemen. Deutsche Wörter sind unregelmäßiger aufgebaut und haben in der Folge lokale und absolute Maxima fast ausschließlich am Anfang der Successor-Werte und am Ende der Predecessor-Werte, so dass fast in jedem Fall der erste und/oder der zweite Buchstabe von vorne wie von hinten als Morphem ausgezeichnet wird. Das führt dazu, dass die wahrscheinlichsten Zerlegungen so gut wie alle aus Ein-Buchstaben-Morphemen bestehen. Da dieses Ergebnis nicht wünschenswert ist, führen für die Analyse der deutschen Texte lediglich ansteigende Successor- bzw. Predecessor-Werte zur Erhöhung des Scores (auf diese Weise wird das Verfahren auch von Benden 2006 umgesetzt). Dadurch werden höchstens zwei Score-Punkte vergeben, weshalb schon einer ausreicht, um eine Morphemgrenze festzulegen. Insgesamt wurden so für die 10.074 analysierten Types 5484 Morpheme ermittelt. Darunter befinden sich alle einzelnen Buchstaben, aber auch relativ lange Zeichenketten wie *charakterisier*, *heimnis* und *furchtsvoll*, aber auch typische Derivationsaffixe wie *-isch*, *-lich*, *-ig*, *-keit* und *-igkeit*. Exemplarisch seien hier die von der Morphemizer-Komponente favorisierten Zerlegungen ausgewählter Wörter, die *-igkeit-* enthalten, aufgeführt:

- Perfekte Zerlegung: *Feind-sel-ig-keit*; *Be-wußt-los-ig-keit*
- Akzeptable Zerlegung: *Un-fähig-keit*; *Ew-ig-keit; Not-wend-igkeit*
- Falsche Zerlegung: *Ge-re-cht-igkeit*; *Ge-walt-tät-igk-eit-en*

Bei geeigneter Korpusgröße und mit der richtigen Konfiguration ist die distributionelle Morphemanalyse in einem gewissen Rahmen auch für natürliche Sprachen anwendbar. Benden (2005) weist darauf hin, dass die Methode vor allem bei agglutinierenden Sprachen mit ihrer insgesamt regelmäßigeren Wortbildung die besten Ergebnisse produzieren sollte.

Diskussion der Ergebnisse

Die Anwendung strukturalistisch motivierter linguistischer Verfahren auf die untersuchten Textsorten deckte weitere Gemeinsamkeiten zwischen VM-Text und P.III-Chiffren auf. Die Wörter beider Chiffren besitzen einen – im Vergleich zu ihren natürlichsprachlichen Pendants – weitaus regelmäßigeren Aufbau, weswegen die eingesetzten Strukturaufdeckungsverfahren schon bei vergleichsweise kleinen Textmengen gute Ergebnisse erzielen.

Die Frage ist nun, inwieweit die gewonnenen Erkenntnisse bei der Rekonstruktion eines

vermuteten VM-Codebuchs dienen können. Die Auswertung der Ergebnisse zur P.III-Chiffre gestalten sich insofern einfach, als dass die Zielstruktur – Trithemius' P.III-Codebuch – bereits vorliegt. Über die Morphemmethode lässt sich ein Großteil der Stämme ermitteln, welche die Zugehörigkeit von Chiffren zu Tabellenspalten bestimmen. Allein damit ist noch nicht viel gewonnen, da der Klartextbuchstabe durch die Zeilenzugehörigkeit codiert wird. Bei der Zuordnung von Chiffren zu Zeilen kann die Auswertung der Graphem-Cluster helfen: In einem der erzeugten Cluster finden sich die Vokale, welche – über die Breite der gesamten Tabelle – immer den gleichen Zeilen zugeordnet sind, in einem anderen Cluster finden sich die Schlusskonsonanten, die in Blöcken von fünf Zeilen immer invariant sind. Basierend auf diesem Wissen ließe sich die kombinatorische Komplexität der Zuordnung einzelner Wörter zu Klartextbuchstaben stark einschränken.

Im Fall des VM gestaltet sich die Auswertung der Ergebnisse ungleich schwieriger, weil die Zielstruktur nicht vorliegt (deren Existenz – man kann es nur wiederholen – darüber hinaus nicht gesichert ist). Dennoch liefert hier vor allem die Morphemzerlegung wichtige Hinweise darauf, wie Stammmorpheme, zuständig für die Zuordnung einer Chiffre zu einer Tabellenspalte, detektiert werden können. In Verbindung mit der Auswertung von Graphem-Clustern könnten die Chiffren auf potentielle Tabellen aufgetragen werden, die sich zur Generierung verschiedener Klartexte nutzen lassen (s.u.).

Die hier durchgeführten Analysen beruhen auf Verfahren, die vom linguistischen Strukturalismus für die Anwendung auf natürlichsprachliche Daten konzipiert wurden. Eine Pointe der vorliegenden Untersuchung ist, dass die Anwendung auf nicht-natürlichsprachliche Daten bessere Ergebnisse generiert als der ursprünglich intendierte Gebrauch. In dieser Hinsicht muss man allerdings anmerken, dass den hier untersuchten NaSp ein flektierender Sprachbau zugrunde liegt. Sprachen mit agglutinierendem Sprachbau (dazu zählen z.B. das Türkische, das Baskische, aber auch viele Plansprachen, z.B. das Esperanto) verfügen über einen sehr viel regelmäßigeren Aufbau ihrer Wörter, so dass die hier eingesetzten Verfahren auf Sprachen dieser sprachtypologischen Klasse wohl besser interpretierbare Ergebnisse erbracht hätten.

6.3 Ausblick: Die Suche nach dem Klartext

Unwillkürlich befällt jeden, der sich mit dem Voynich-Manuskript auseinandersetzt, das Verlangen, das Rätsel zu lösen, welches in ihm verborgen liegt. Die hier durchgeführten Analysen mögen einen kleinen Teil zu dieser Lösung beitragen: Über den Vergleich statis-

tischer Kennwerte von VM-Text, P.III-Chiffre und NaSp-Vergleichstexten konnte gezeigt werden, dass die Möglichkeit besteht, dass der VM-Text durch ein Verschlüsselungsverfahren erzeugt wurde, welches auf ähnliche Weise wie das P.III-Verfahren funktioniert. Weiterhin konnten durch die Adaption linguistisch-strukturalistischer Verfahren für die Analyse von künstlich erzeugten (d.h. nicht-natürlichsprachlichen) Wörtern erste Ansätze zum Entwurf von Tabellen skizziert werden, in welche diese Wörter in systematischer Weise aufgetragen werden könnten. Im P.III-Verfahren bilden solche Tabellen von Ersetzungschiffren den Schlüssel zur Rekonstruktion eines mit P.III verschlüsselten Klartexts. Die Ähnlichkeit des VM zur P.III nährt die Hoffnung, dass es auf ähnliche Weise entschlüsselt werden könnte.

Nun lehrt aber die Geschichte der Forschungen zum VM, dass diejenigen, die sich mit ihm auseinandersetzten, oft zu nicht zutreffenden Schlussfolgerungen verleitet wurden. Am treffendsten wird dies vielleicht von Kennedy & Churchill (2004:282) formuliert:

> "Die 'Kunst' der Voynich-Handschrift besteht darin, jedem Forscher den Spiegel vorzuhalten, so dass er anstelle der Wahrheit, die der Handschrift zugrunde liegt, ein Abbild der eigenen Vorurteile und Überzeugungen erblickt"

Die Gefahr, in einen Zerrspiegel zu blicken, ist im Fall dieses seltsamen Manuskriptes immer gegeben. Aufgrund dessen wird an dieser Stelle kein *direkter* Versuch unternommen, die bisher gewonnenen Erkenntnisse *vor* einer kritischen Analyse durch die wissenschaftliche Öffentlichkeit zu einem Angriff auf den Klartext des VM zu nutzen.

Die Dechiffrierung einer so außergewöhnlichen Chiffre, bei welcher der Verschlüsseler über eine große Auswahl von verwendbaren Geheimtexteinheiten pro Klartexteinheit verfügt, wäre ein sehr aufwendiges und mühsames Unternehmen. Ein solches ist im Kontext dieser Arbeit kaum zu leisten und sollte besser durch wirkliche Experten für die Kryptoanalyse von Texten angegangen werden. In der Wissenschaft ist es aber nicht unüblich, dass Spezialisten bestimmte Teilprobleme lösen, so dass Spezialisten anderer Fachrichtungen darauf aufbauen können. Nicht zuletzt wurde Tesla auch in Hinsicht auf eine solche kollaborative Arbeitsweise hin angelegt: Sämtliche durchgeführten Analysen sind dokumentiert und als Tesla-Experimente veröffentlicht, so dass sie in jeder Tesla-Instanz geprüft und modifiziert werden sowie als Grundlage für eigene Ansätze (z.B. für die Ermittlung des Klartextes) dienen können. Damit können einerseits die hier vertretenen Ansichten, dass VM-Wörter möglicherweise als Substitute für Klartext*buchstaben* aufzufassen sind und dass es für jeden Klartextbuchstaben eine Reihe von Wort-Homophonen gibt, überprüft werden. Andererseits lassen sich aber auch die ermittelten Graphemcluster und die Zer-

legung der VM-Wörter in Morpheme wiederholen und durch konfigurationelle Veränderungen der Experimente modifizieren. Darauf aufbauend ließen sich Hypothesen darüber aufstellen, welche VM-Wörter möglicherweise als Homophone genutzt wurden. Gelänge es, eine kritische Masse von potentiellen Homophonen zu identifizieren, würde dadurch ein Angriff auf die VM-Chiffre über Häufigkeitsanalysen möglich werden.

Anstatt aber hier einen direkten Versuch zu unternehmen, den VM-Klartext anzugreifen, soll das allgemeine Problem der Kryptoanalyse von Texten anhand einer charakteristischen Herangehensweise beim Angriff auf Geheimtexte demonstriert werden, die sich dazu eignet, eine weitere zentrale Eigenschaft von Tesla zu veranschaulichen: In 3.2.1 wurde darauf eingegangen, dass die Ein- und Ausgabe-Schnittstellen von Tesla-Komponenten im Gegensatz zu denen vergleichbarer Komponentensysteme wie *Gate* und *UIMA* weitgehend unrestringiert sind.[51] Dadurch sind Tesla-Komponenten in der Lage, sehr aussagekräftige und vielschichtige Ergebnisse zu liefern, die von anderen Komponenten konsumiert werden können, womit komplexe, nichtsdestrotz modular organisierte Analysen innerhalb abgeschlossener Tesla-Experimente realisiert werden können. Diese Eigenschaft von Tesla konnte durch die bisherigen Analysen noch nicht umfassend dargestellt werden. Die beteiligten Komponenten, deren Output weiter konsumiert wurde, schrieben entweder tokenbasierte Annotationen (wie z.B. der Präprozessor oder die Substitutionskomponente) oder erzeugten einfache Merkmalsvektoren (wie der Graphemizer), die als Input für eine Cluster-Komponente dienten. Dass eine Tesla-Komponente komplexe Objekte als Ergebnis produzieren kann, welche von weiteren Komponenten als Grundlage ihrer Berechnungen genutzt werden können, wird deshalb im Folgenden an einem Anwendungsfall, der in den Bereich der Kryptoanalyse zur Rekonstruktion von Klartexten passt, demonstriert.

Basis für das vorgestellte Verfahren ist die Beobachtung, dass natürliche Sprache "ein schwer ausrottbares Gerüst von Gesetzmäßigkeiten" enthält (Bauer 2000:235), so insbesondere Wiederholungsmuster. Die Detektion solcher wiederholten Abfolgen ist deshalb eine gute Grundlage für vielversprechende Ansätze bei der Klartextsuche (vgl. Bauer 2000:235ff). Bei einer monoalphabetischen Verschlüsselung verhalten sich Wiederholungsmuster invariant, auch bei einer polyalphabetischen Verschlüsselung mit kurzer Schlüssellänge sind sie bei ausreichender Textgröße detektierbar. Die Verwendung von Homophonen bei der Chiffrierung zerstört auch einen Großteil der Wiederholungsmuster des

51 Dies gilt – wie in 3.2.2 gezeigt – in *globaler*, d.h. das gesamte Komponentensystem betreffender Hinsicht. Die Kompatibilität von Komponenten wird durch *lokale* Restriktionen der von der Komponente ausgefüllten Rolle aus dem Tesla-Rollensystem (TRS) sichergestellt.

Klartextes, dennoch können einige von ihnen erhalten bleiben, beispielsweise durch eine immer gleiche Abfolge im Auswahlprozess. In den anschließenden Abschnitten wird ein Verfahren vorgestellt, das die Detektion von Wortwiederholungen in gegebenen Texten innerhalb von Tesla realisiert. Dabei werden zwei Komponenten vorgestellt, die über ihre Schnittstellen ein komplexes Datenobjekt – einen n-Gramm-Baum – austauschen.

6.3.1 Problemstellung: Suche nach Mustern im Text

Die spezifischen Gesetzmäßigkeiten *innerhalb* von Wörtern des VM wurden bereits in 6.2 thematisiert, nun werden charakteristische Eigenschaften von Wort*kombinationen* analysiert. Gehäuft gemeinsam auftretende Wörter bezeichnet man in der Linguistik als *Kollokationen*. Für den Fall, dass man wie wir das gemeinsame Auftreten direkt benachbarter Wörter analysieren will, bietet sich ein wortbasiertes *n-Gramm-Modell* an. Bei einem solchen n-Gramm-Modell wird der Text in Fragmente der Länge n zerlegt. Fragmente der Länge eins (in unserem Fall einzelne VM-Wörter) bezeichnet man als Monogramme, solche der Länge zwei als Bigramme, usw. Zusammengefasst werden n-Gramme auch als Multigramme bezeichnet. Neben der Kryptoanalyse (vgl. Bauer 2000:235ff) werden auch in der Linguistik Statistiken über die Häufigkeiten von Multigrammen ausgewertet, etwa um die einem Text zugrundeliegende Sprache (Dunning 1994) oder Mehrwortlemmata[52] zu bestimmen. Manning & Schütze (1999:152ff) beispielsweise ermitteln signifikantes gemeinsames Auftreten von Wörtern durch sogenannte Signifikanztests, bei denen das Vorkommen von Wortkombinationen in Relation zur Häufigkeit der einzelnen, an der Kombination beteiligten Wörtern gesetzt wird. Dazu müssen sowohl die Häufigkeiten der einzelnen Wörter, als auch die aller n-Gramme ermittelt werden. Ein solches Verfahren ist nur für kleine n effektiv umzusetzen (vgl. Manning & Schütze 1999:194, die Autoren beschränken sich auf die Ermittlung von Bi- und Trigrammen), da die Zahl der möglichen n-Gramme exponentiell ansteigt ($|nGrams| \approx Types^n$).

Im Kontext der Kryptoanalyse sind wir aber konkret an denjenigen Wort-Multigrammen interessiert, die aus möglichst vielen Elementen bestehen (Maximierung der Länge), und dabei möglichst häufig vorkommen (Maximierung der Anzahl).[53] Es ist also nicht ausrei-

52 Mehrwortlemmata werden auch als *usuelle Wortverbindungen* bezeichnet. Beispiele sind Redewendungen wie "*Mit Kind und Kegel*" oder "*eine Hand voll*", aber auch festgelegte Wortkombinationen wie "*Zähne putzen*".

53 Länge und Anzahl können nicht gemeinsam maximiert werden, da sie sich reziprok proportional verhalten: Das längste Multigramm n_{max} ist immer der gesamte Text, es kommt selbstredend auch nur einmal im Text vor ($m = 1$). Für jedes Multigram mit n_x gilt, dass seine Häufigkeit kleiner oder

chend, lediglich alle Bi- und Trigramme über die Wörter des Textes zu ermitteln.

Wir nutzen deshalb eine Datenstruktur, welche in der Lage ist, die Abfolge von Einheiten in einem Text auf eine platzsparende, aber verlustfreie Weise abzubilden. Diese Datenstruktur ist ein *Suffixbaum*, der bereits für die effiziente Lösung einer Vielzahl von Problemen aus dem Bereich der Textprozessierung genutzt wird.[54] Bildet man einen Text S auf einem Suffixbaum T ab, so enthält T alle Suffixe (Endungen) des Textes, wobei mehrfach auftretende Teilstrings von S nur einmal in T enthalten sind. Es können auch mehrere Texte $S_1 \ldots S_n$ auf T abgebildet werden; in diesem Fall nennt man T einen *generalisierten Suffixbaum*[55]. Abbildung 6.4 zeigt einen solchen generalisierten Suffixbaum, der sich aus zwei Zeilen des VM-Textes ergibt.[56] Da ein Suffixbaum alle Suffixe eines Textes enthält, enthält er auch alle n-Gramme dieses Textes. Den Umstand können wir ausnutzen, um unsere Suche nach den längsten n-Grammen zu bestreiten, die mindestens in einer bestimmten Häufigkeit vorkommen.

6.3.2 Hypothesen

Auch wenn für die Verschlüsselung jedes Buchstabens eine Reihe von Homophonen zur Verfügung stand, sollte es während der Codierung vorgekommen sein, dass identische Buchstabenfolgen durch eine Sequenz identischer Chiffren, anders ausgedrückt durch Muster, ersetzt worden sind. Durch eine Analyse über die Häufigkeit von n-Grammen lassen sich diese Muster aus dem Text extrahieren und – sofern sie in ausreichender Zahl vorliegen – für einen kryptoanalytischen Angriff auf die Chiffre nutzen.

6.3.3 Design des Experiments

Um Wiederholungsmuster im VM-Text zu finden, wird dieser zunächst wieder von einem Reader eingelesen. Der Suffixbaum wird über Wörter aufgebaut, daher muss der Text tokenisiert werden. Man kann nun entweder den Gesamttext als eine Sequenz in den Suffixbaum einspeisen, oder man ermittelt Teilsequenzen, so dass ein generalisierter

gleich der Häufigkeit aller enthaltenen Multigramme (n_{x-y}) ist.

54 Eine grundlegende Einführung in den Aufbau, die Erstellung und die Anwendung von Suffixbäumen findet sich bei Gusfield (1997).

55 Zum Unterschied zwischen dem hier verwendeten Suffixbaum (engl. SuffixTree) und dem oben (6.2.1) eingesetzten KeywordTree vgl. Fußnote 37.

56 Das Konzept eines Suffixbaums lässt sich umso besser darstellen, je redundanter der Text ist. Deswegen wurden hier zwei Textzeilen ausgewählt, welche das Mittel der mehrfachen Wortwiederholung gebrauchen.

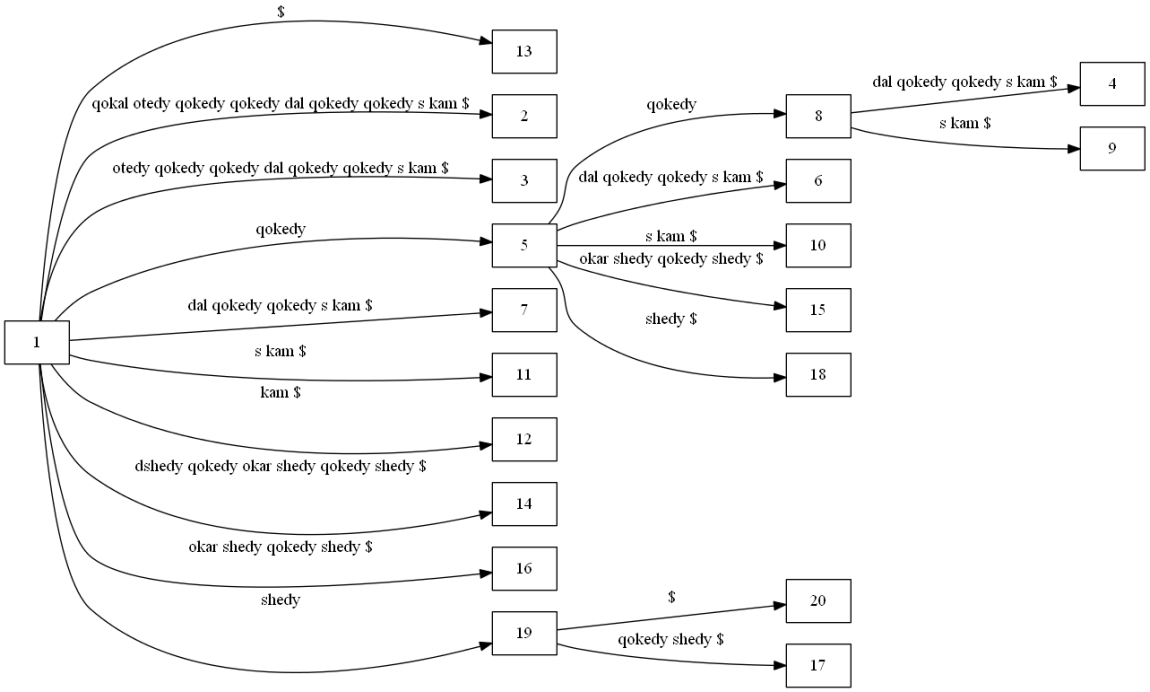


Abbildung 6.4: *Visualisierung eines generaliserten Suffixbaums, der aus zwei Zeilen des VM-Textes generiert wurde (f78r, Zeilen 5 "qokal otedy qokedy qokedy dal qokedy qokedy s kam" und 6 "dshedy qokedy okar shedy qokedy shedy". Die Beschriftung befindet sich jeweils über den Kanten. "qokedy", "shedy" sowie das Wortpaar "qokedy qokedy" kommen mehrfach im Baum vor und haben daher mehrere Nachfolgerknoten.*

Suffixbaum über einzelne Textstücke generiert werden kann. Letztere Vorgehensweise hat den Vorteil, dass die Tiefe des Baums maximal der längsten Teilsequenz entspricht. Im Fall des VM-Textes bieten sich Zeilen oder Paragraphen als solche Teilsequenzen an. Diese werden ebenfalls während der Präprozessierung ermittelt.

Die Sequenzen werden dann zusammen mit den Wörtern an die Komponente übergeben, die zuständig ist für den Aufbau des generalisierten Suffixbaums (in der Abbildung 6.5 der `NGramTreeGenerator`). Der erzeugte Baum ist wiederum Grundlage für die Komponente, in der die Mustersuche realisiert wird (in der Abbildung der `PatternDetector`). In dieser Komponente kann festgelegt werden, was Mindesthäufigkeit und Mindestlänge derjenigen Muster sein soll, die zurück auf den Ausgangstext abgebildet werden.

6.3.4 Benötigte Komponenten

Die zum Einsatz kommenden Komponenten Reader (`VoynichInterlinearArchiveReader`) und Präprozessor (`SPre`) wurden bereits in den vorangegangenen Analysen vorgestellt,

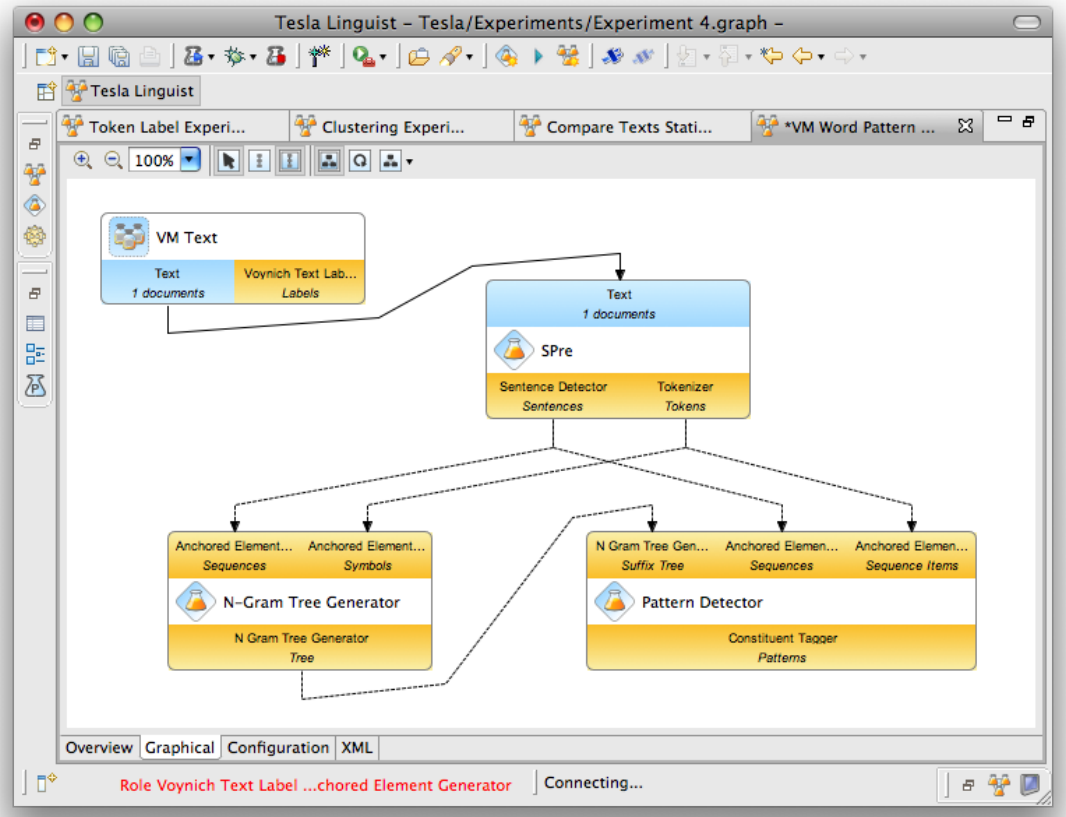


Abbildung 6.5: *Graphischer View der Muster-Detektions-Experimente. VM-Reader und SPre werden konfiguriert, als Sequenzen können entweder die Zeilen (Reader) oder die Paragraphen (SPre) gewählt werden.*

dabei wurde auch auf die Fähigkeit des Readers eingegangen, nur bestimmte, über die Konfiguration auswählbare Abschnitte des VM-Textes zu liefern, um einzelne Sektionen oder Currier-Dialekte getrennt voneinander analysieren zu können. Darüber hinaus werden noch zwei weitere Komponenten benötigt:

Suffixbaum-Generator

Die Erstellung eines Suffixbaums wird von der Komponente `NGramTreeGenerator` übernommen.[57] Diese benötigt als Input zum einen Sequenzen von Einheiten (etwa ganze

57 Die Komponente heißt nicht SuffixTreeGenerator, weil sich die maximale Tiefe des Baums festlegen lässt. Insofern enthält der Baum u.U. nicht alle Suffixe des eingespeisten Textes, sondern lediglich alle n-Gramme bis zur eingestellten Maximaltiefe.

Texte, Paragraphen, Sätze oder Zeilen), zum anderen die Einheiten selbst (Wörter, u.U. auch Zeichen). Der Generator bildet daraus einen generalisierten Suffixbaum, bei dem die Einheiten allerdings nicht den Kanten, sondern den Knoten selbst zugeordnet sind.[58] Jeder Knoten hat Referenzen zu seinem eindeutigen Vorgängerknoten und, sofern vorhanden, zu seinen Nachfolgerknoten im Baum. Zusätzlich wird zu jedem Knotenobjekt protokolliert, welche Tokens des Ausgangstextes durch den Knoten repräsentiert werden. Dies ist notwendig, um die n-Gramme des Baumes wieder auf den Ursprungstext zurückführen zu können. Zudem wird auf diese Weise vermerkt, wie oft jeder Knoten des Baumes durchlaufen wurde.

Über die Konfigurationsschnittstelle wird festgelegt, ob die Sequenzen vorwärts oder rückwärts in den Baum eingespeist werden sollen. Hierdurch kann festgelegt werden, ob man einen Suffix- oder einen Präfix-Baum generiert. Außerdem kann die Maximaltiefe des Baumes angegeben werden – wie oben erwähnt, sprengen Bäume höherer Ordnung schnell den zur Verfügung stehenden Speicherplatz, vor allem bei der Analyse größerer Korpora.

Die Ausgabe der Komponente ist über die gleichnamige Rolle (`NGramTreeGenerator`) festgelegt und besteht aus einem `NGramTree`-Objekt. Der Tree bietet Zugriff auf seinen Wurzelknoten, der auf seine Nachfolgerknoten verweist, welche dann rekursiv[59] durchlaufen werden können.

Muster-Identifizierer

Die Identifizierung von Mustern übernimmt die Komponente `PatternDetector`, welche die Rolle `NGramTreeGenerator` konsumiert und damit Zugriff auf einen `NGramTree` erhält, der nach den längsten bzw. häufigsten Mustern durchsucht werden soll. Zusätzlich müssen noch die Sequenzen und die Einheiten, aus denen der `NGramTree` generiert wurde, spezifiziert werden, da man diese benötigt, um die ermittelten Muster zurück auf den Ausgangstext abzubilden.

Über die Konfiguration wird festgelegt, was die Mindestlänge der zu ermittelnden Muster (*minlength*) sein soll und mit welcher Häufigkeit sie mindestens aufzutreten haben, um berücksichtigt zu werden (*minfreq*).

58 Dieses Design kommt der Implementation in einer objektorientierten Programmiersprache entgegen, da man die Kanten so nicht als eigene Objekte anlegen muss, sondern diese einfach durch Referenzen auf Knoten-Objekte realisieren kann.

59 Rekursion ist der "Bezug einer Datenstruktur oder einer Funktion innerhalb ihrer Definition auf sich selbst" Schwiebert (2011: Kap. 2.1.2), vgl. ebd. für ein Beispiel der wechselseitigen, endlosen Rekursion.

Die von der Komponente ermittelten Muster werden als Annotationen an alle Bereiche des Ausgangstextes geschrieben, in denen sie vorkommen. Da es sich um unterbrechungsfreie Sequenzen von Einheiten handelt, wurde dafür eine bestehende Rolle – die des `Constituent Taggers` – wiederverwendet, welche derartige Annotationsobjekte als Output spezifiziert.

6.3.5 Ergebnisse

Die Durchführung der Experimente mit unterschiedlichen Konfigurationen ergab eine erstaunlich geringe Anzahl von Wort-Wiederholungsmustern im VM-Text. Die längsten Muster werden von Einzelbuchstabenkombinationen gebildet, die in der astrologischen Sektion des VM auf die konzentrischen Kreise aufgetragen sind – auf *f57v* etwa findet sich viermal das Muster *o-l-d-r-v-x-k*. Abgesehen davon gibt es nur zwei 4-Gramme, die überhaupt mehrfach auftreten: *shedy.qol.shedy.qokaiin* und *ol.shedy.qokedy.qokeedy* (je zwei Vorkommen). Trigramme, die mindestens zweimal auftreten, werden in 413 Fällen annotiert, davon kommt allerdings nur eines (*ol.shedy.qokedy*) fünfmal, zwei weitere (*ol.s.aiin* und *chey.qol.chedy*) viermal vor. Screenshot 6.6 zeigt mittels des Tesla-Visualisierers für farblich unterlegten Text einen Ausschnitt von *f84r*. An dieser Stelle herrscht die größte Dichte von Trigrammen, die mindestens dreimal auftreten.

In einem weiteren Experiment wurde der `PatternDetector` so konfiguriert, dass alle Multigramme ausgezeichnet wurden, die mindestens fünfmal im Text auftreten. Insgesamt wurden mehr als 2500 Annotationen geschrieben.[60] Die Ergebnisse des Experiments sind ausschnittsweise in Abbildung 6.7 visualisiert.

Im Vergleich zu den natürlichsprachlichen Vergleichstexten, die mit der gleichen Methode untersucht wurden, finden sich sehr wenige Wortwiederholungen im VM-Text. Das ist insofern erstaunlich, als dass man eigentlich bei einem Text, der über Ersetzungstabellen für Buchstaben konstruiert wurde (nach der in dieser Arbeit vertretenen P.III-Hypothese) oder der alternativ dazu aus einer sinnlosen Aneinanderreihung von Phantasiewörtern besteht (nach der Hoax-Hypothese) davon ausgehen sollte, dass Wiederholungsmuster in größerer Zahl auftreten. Offenbar ist es bei der Anfertigung des Manuskripts aber gelungen, durch ein spezifisches Auswahlverfahren über die Wörter des Textes solche Muster weitgehend zu vermeiden.

Für einen kryptoanalytischen Angriff erscheint die Zahl der gefundenen Muster zu gering.

60 Mehr als zwei Elemente überspannen nur die fünf Annotationen über das oben erwähnte Trigramm *ol.shedy.qokedy*. Nicht berücksichtigt wurden wieder Multigramme, die aus Ketten von Einbuchstaben-Einheiten bestehen.

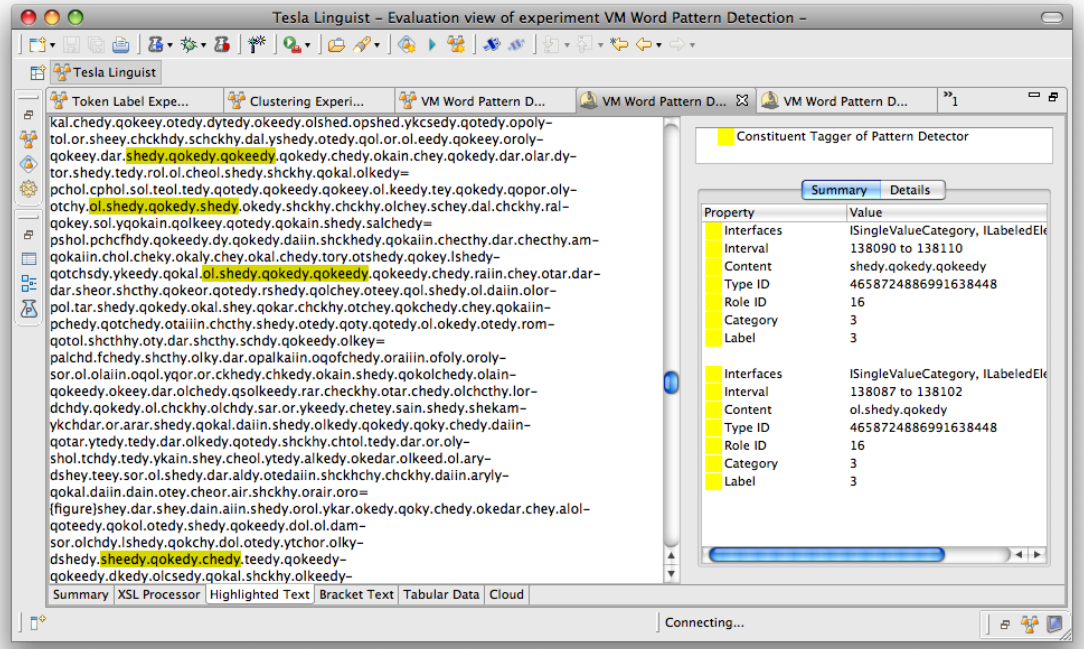


Abbildung 6.6: *Tesla-Visualisierung von detektierten Wiederholungsmustern im Text. Zu sehen ist ein Ausschnitt des Textes von Folio 84r (balneologische Sektion), mit den Resultaten des PatternDetectors (Konfiguration: minlength=3, minfreq=3). Die Sequenz* ol.shedy.qokedy.qokeedy *ist zwar als Ganzes markiert, wie man in der Detaildarstellung rechts im Bild sieht, handelt es sich nicht um ein 4-Gramm (das nur zweimal vorkommt, siehe oben), sondern um zwei sich überlappende Trigramme, die fünf bzw. dreimal auftreten. Insgesamt wurden von der Komponente 137 Annotationen geschrieben, davon allerdings nur 45 Trigramme, die nicht ausschließlich aus Einzelbuchstaben oder nicht erkannten Zeichen bestehen. Die Annotationen sind sehr viel spärlicher über den Text verteilt, als dieser Screenshot suggeriert.*

Eine Möglichkeit, die Ergebnisse zu verbessern, wäre, die starre Analyse von n-Grammen aufzubrechen, indem Abweichungen innerhalb von Sequenzen zugelassen werden. Dies erreicht man über sogenannte weiche Matchverfahren, z.B. durch die Verwendung von Wildcards oder die k-mismatch-Methode (vgl. Gusfield 1997:196ff). Diese Verfahren ließen sich auch auf Suffixbäume anwenden, die Zeichen- anstatt Wort-basiert erstellt werden, um wiederholte Abfolgen ähnlicher Wörter aufzudecken, die Rückschlüsse auf den Auswahlprozess zulassen könnten.

Hier sollte aber vor allem veranschaulicht werden, wie Tesla-Komponenten über komplexe Objekte – in diesem Fall über einen Suffix-Baum – miteinander kommunizieren können. Man mag einwenden, dass `NGramTreeGenerator` und `PatternDetector` in einer einzigen

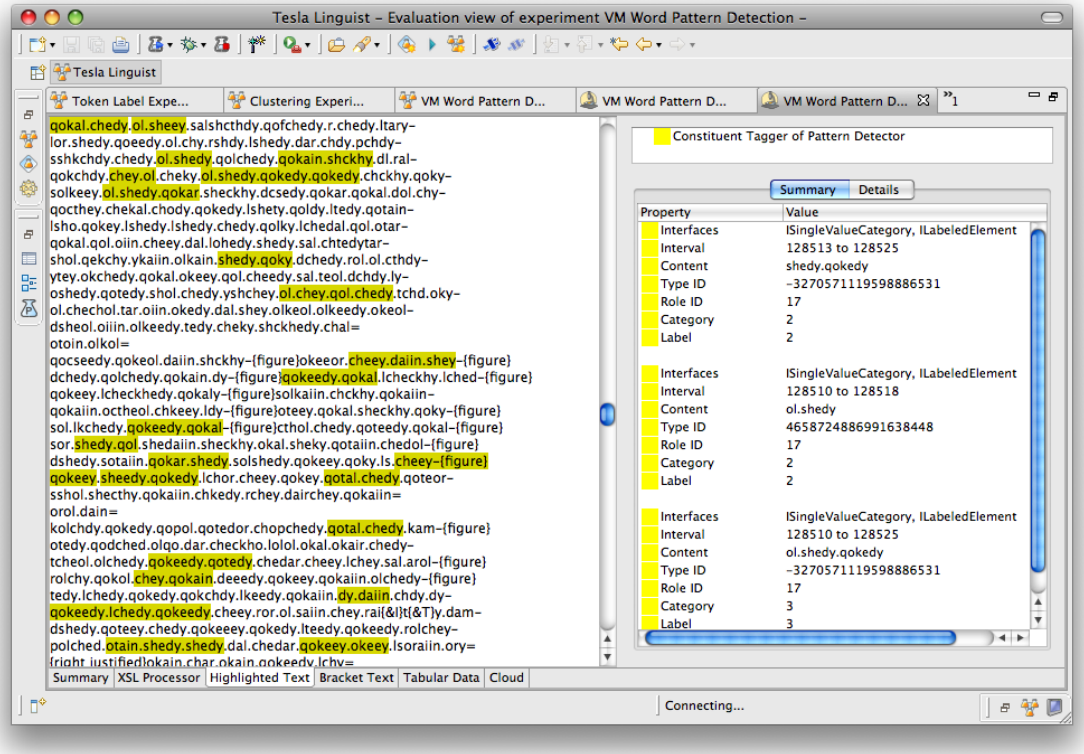


Abbildung 6.7: *Visualisierung der Annotationen des Pattern Detectors (Konfiguration: minlength=2, minfreq=5) für einen Textausschnitt von f81r (ebenfalls balneologische Sektion). Die insgesamt mehr als 2500 Annotationen sind relativ gleichmäßig über den gesamten Text verteilt, insofern ist der Screenshot (im Gegensatz zu dem in Abbildung 6.6) als repräsentativ anzusehen.*

Komponente hätten realisiert werden können. Tatsächlich benötigt der `PatternDetector`, um seine Arbeit durchführen zu können, ein `NGramTree`-Objekt, das exakt so aussehen muss, wie es der Generator liefert. Umgekehrt besteht jedoch kein Abhängigkeitsverhältnis: Der `NGramTreeGenerator` wurde im Rahmen der Arbeit von Schwiebert (2011) entworfen, wo er – unabhängig von dem für die vorliegende Arbeit entworfenen `PatternDetector` – für die Identifikation von Syntagmen und Paradigmen in natürlichsprachlichen Texten eingesetzt wird. Diese Wiederverwertung wird erst durch die modulare Anlage der Komponenten ermöglicht.

Kapitel 7

Fazit und Ausblick

Zielsetzung dieser Arbeit war, die Motivation für die Entwicklung des Text Engineering Software Laboratory aus den Anforderungen aktueller Ansätze der wissenschaftlichen Arbeitsweise im Bereich der Verarbeitung von Texten herzuleiten (1), die grundlegenden Konzepte dieses Softwaresystems in kompakter Form darzustellen (2) und schließlich seine Anwendung zu demonstrieren, indem es für die Bearbeitung einer exemplarischen wissenschaftlichen Fragestellung eingesetzt wird (3).

Dies wurde im Verlauf der Arbeit wie folgt umgesetzt: (1) Im Kapitel 2 "Textprozessierung" wurde herausgestellt, dass Wissenschaft vor allem auf dem Austausch von Informationen beruht. Der Rahmen der Möglichkeiten für einen solchen Informationsaustausch wird durch den Einsatz moderner Informations-, Kommunikations- und Softwaretechnologien massiv erweitert. Im besonderen Maße können Forscher davon profitieren, die auf der Basis von textuellen Daten arbeiten, denn ihr Untersuchungsgegenstand lässt sich ohne weiteres virtuell abbilden, verarbeiten und distribuieren. Der offene Textbegriff von Texten als Sequenzen diskreter Einheiten ermöglicht es, für Wissenschaftler aus Teilbereichen ganz unterschiedlicher Zweige der Wissenschaft eine gemeinsame Basis – die der Textprozessierung – zu identifizieren. Damit auf diesem Fundament die Potentiale ausgeschöpft werden können, welche durch den heutigen Stand der Technik geboten werden, wird ein Werkzeug in Form einer auf die spezifischen Bedürfnisse der Textprozessierung zugeschnittenen Software benötigt. Das Werkzeug muss offen sein für die Verarbeitung sämtlicher Formate textueller Daten, es muss ohne Kenntnisse einer höheren Programmiersprache bedienbar sein und es muss schließlich in der Lage sein, die experimentellen Abläufe, die in ihm durchgeführt werden, lückenlos zu dokumentieren. (2) Das Kapitel 3 "Tesla" beschreibt das im Rahmen dieser Dissertation gemeinsam mit Stephan Schwiebert entwickelte gleichnamige Softwaresytem als Instanz eines solchen Werkzeugs. Tesla ist als virtuelles Labor konzipiert, das in der Form eines Client-Server-basierten offenen Komponentensystems umgesetzt wurde. Es erlaubt die Prozessierung unterschiedlicher Textfor-

mate, realisiert Experimente über Ausgangsdaten durch die Verknüpfung konfigurierbarer Komponenten in einem graphischen Editor und protokolliert den gesamten Aufbau sowie die Resultate der durchgeführten Analysen. (3) Im weiteren Verlauf der Arbeit wurde ein konkretes Problem der Textprozessierung als Anwendungsfall eingeführt (Kapitel 4 "Das Voynich-Manuskript" und Kapitel 5 "Kryptographie der frühen Neuzeit"), das schließlich im Kapitel 6 "Analysen" innerhalb von Tesla behandelt wurde. Hierbei stand im Vordergrund, die Einsatzmöglichkeiten von Tesla an möglichst vielen unterschiedlichen und leicht verständlichen Szenarien zu veranschaulichen. Innerhalb der konkreten Anwendungen wurde regelmäßig auf die Vorzüge eingegangen, die sich durch den Gebrauch von Tesla ergeben und die letztlich auch die Motive für die Entwicklung dieser Software waren. Sie sind in der folgenden Aufstellung aufgeführt und den vier Leitgedanken, welche grundlegend für die Entwicklung von Tesla waren, zugeordnet: Bedienbarkeit, Funktionalität, Offenheit und Reproduzierbarkeit.

Bedienbarkeit: Der Tesla-Client wurde als Eclipse-Plugin realisiert, so dass auf die umfangreiche Funktionalität dieser Plattform zurückgegriffen werden kann. Der Client bietet eine homogene Arbeitsumgebung, die durch das Eclipse-Konzept, unterschiedliche Perspektiven anzubieten, von verschiedenen Benutzergruppen verwendet werden kann: So können Anwender der textprozessierenden Werkzeuge (über die *Linguist Perspective*) die Plattform genauso nutzen wie Entwickler, die neue textprozessierende Werkzeuge implementieren (über die *Developer Perspective*). Durch die Offenheit des Codes hat der Anwender jederzeit die Möglichkeit, die konkreten Implementierungen seiner genutzten Komponenten zu analysieren, evtl. selbst zu modifizieren oder gleich gegen eine eigene Realisierung auszutauschen. Diese Art des *Programmierens im Kleinen* ist aber nicht zwingend notwendig für die Nutzung von Tesla. Vielmehr gestattet das Komponentenkonzept dem Anwender auch ein *Programmieren im Großen*. Dabei muss er sich nicht mit konkreten Implementierungsdetails der von ihm genutzten Werkzeuge auseinandersetzen, sondern kann diese einfach in der graphischen Benutzeroberfläche nach Bedarf miteinander kombinieren und einzeln konfigurieren.

Funktionalität: Auf der einen Seite bietet Tesla Grundfunktionalitäten, die zum Arbeiten mit Texten und Komponenten benötigt werden, im Tesla-Client sind dies etwa der Korpusmanager für die Verwaltung von Texten, ein graphischer Editor für den Entwurf und die Konfiguration von Versuchsaufbauten, daneben ein virtueller Server, der sich um die Abarbeitung der Experimente und die Datenhaltung kümmert, sowie unterschiedliche Visualisierer für die Ergebnisse der durchgeführten Analysen. Auf der anderen Seite hält

Tesla bereits ein beachtliches Inventar an textprozessierenden Komponenten bereit, die verwendet werden können, um den Inhalt verschiedenartiger Textformate einzulesen, Metadaten und vorhandene Annotationen zu interpretieren, auf unterschiedlichen Ebenen (Zeichen, Wort, Satz, Paragraph) zu tokenisieren, tokenisierte Einheiten auszuzeichnen (PoS-Tags, Gazetteer-Kategorien), Merkmale aus Texten zu extrahieren (statistische oder auf Kookkurenzen basierende Vektor-Generatoren), in Strukturen zu aggregieren (Suffix- und Keyword-Trees, Cluster) und diese Strukturen wiederum zu analysieren (Musterdetektion). Mit jedem Unternehmen, zu dessen Durchführung Tesla beitragen kann, wird der Umfang dieses Inventars zunehmen und weiteren Forschungen zur Verfügung stehen.

Offenheit: Prinzipiell lassen sich alle Arten von sequenziell organisierten Daten innerhalb von Tesla verarbeiten. Ein solch offener Ansatz erlaubt es Wissenschaftlern unterschiedlicher Forschungsrichtungen, dieses System zu nutzen. Im Zuge dessen wird eine Möglichkeit des Austausch von Werkzeugen und Verfahren zwischen diesen Forschungszweigen mit ihren unterschiedlichen theoretischen Ausrichtungen ermöglicht. In dieser Arbeit wurden Methoden, die ursprünglich für so unterschiedliche Bereiche wie die Korpuslinguistik (korpusstatistische Kennwerte), die Kryptoanalyse (Koinzidenzwerte), die Bioinformatik (Random Walk, Suffixbäume) und eine Form des linguistischen Strukturalismus (Minimalpaaranalyse, distributionelle Morphemanalyse) entwickelt worden sind, zur Bestätigung der Ähnlichkeit von Texten, zur Klassifikation linguistischer Einheiten (von unbekannten Wörtern einer Phantasiesprache) sowie zur Identifikation von Mustern innerhalb von Texten genutzt. Die gegenseitige Bereicherung über bislang relativ schwer überwindbare Grenzen zwischen den Einzelwissenschaften wird unterstützt durch die Granularität in der Anlage einzelner Komponenten, die damit in unterschiedlichen Anforderungsszenarien, aber dennoch innerhalb einer homogenen Arbeitsumgebung eingesetzt werden können.

Reproduzierbarkeit: Die Tatsache, dass das Tesla-Komponentenmodell relativ unrestrigierte Ein- und Ausgabeschnittstellen für Komponenten zulässt, hat einen weiteren gewichtigen Vorteil, da hierdurch alle benötigten Aufgaben der Textprozessierung (Vorprozessierung, Merkmalserstellung, Merkmalsauswahl, Clustering, Klassifikation, Aggregation in komplexen Strukturen, Mustersuche etc.) innerhalb von in sich geschlossenen Experimenten durchgeführt werden können. Daraus resultiert die Möglichkeit, die geleistete Analysearbeit durchgängig und vollständig zu dokumentieren. Die Dokumentation liegt in einem Format vor, das in jede Tesla-Instanz zu Zwecken der Überprüfung oder Modifikation importiert werden kann. Der Quellcode aller beteiligten Komponenten steht unter einer OpenSource-Lizenz, so dass auch - so gewünscht - die algorithmische Basis der

Verfahrensweise nachvollzogen werden kann. Die Ergebnisse wissenschaftlicher Tätigkeit werden damit genauso weitergebbar wie die Bedingungen ihrer Erzeugung.

Tesla wurde als System konzipiert, das es ermöglicht, eine große Bandbreite von Fragestellungen aus dem Bereich der Textprozessierung zu bearbeiten. Naturgemäß konnte hier nur ein kleiner Teil der möglichen Anwendungsbereiche skizziert werden. Zur Zeit wird Tesla in drei weiteren Projekten eingesetzt. Eines davon ist das im Laufe dieser Arbeit schon mehrfach erwähnte Dissertationsprojekt von Schwiebert (2011), das neben einer ausführlichen Beschreibung der technischen Grundlagen von Tesla einen Anwendungsfall beschreibt, in dem Tesla eingesetzt wird, um syntagmatische und paradigmatische Relationen aus natürlichsprachlichen Daten zu extrahieren. Ein weiteres laufendes Dissertationsprojekt verwendet Tesla zur Modellierung des Prozesses der Bedeutungskonstitution in natürlichsprachlichen Texten. Weiterhin dient Tesla auch als zentrales Labor für die Durchführung des Projektes "Text – Information – Wissen" (gefördert von der *RheinEnergieStiftung Wissenschaft*), bei dem ein Ansatz zur Informationsextraktion auf Grundlage der Detektion und Relationierung von Mustern in Texten in Kooperation von Sprachlicher Informationsverarbeitung (Prof. Dr. Jürgen Rolshoven) und der Kölner Bioinformatik (Prof. Dr. Thomas Wiehe) ausgearbeitet wird. Ferner wird Tesla auch als Tool in der Lehre (im Studiengang Informationsverarbeitung) eingesetzt, wo es im Rahmen von Hauptseminar-, Bachelor- oder Masterarbeiten als Basis für Komponentenautoren und für empirische Studien in der Fachrichtung Computerlinguistik bzw. Sprachtechnologie dient.

Diese Arbeit schafft eine Grundlage für weiterführende Forschungen unter Einsatz von Tesla auch außerhalb der Institutsgrenzen, innerhalb derer es entstanden ist. Unterstützt werden soll dies zum einen durch die Implementierung weiterer Funktionalität in Form von Tesla-Komponenten, zum anderen durch die Integration von Tesla in bestehende Forschungsinfrastrukturen, um deren Nutzern zusätzliche Funktionalität in ihrer gewohnten Arbeitsumgebung anzubieten. Konkret ist bisher die Zusammenführung des Tesla-Clients und des *TextGrid*-Clients geplant. Vereinfacht wird dieses Vorhaben durch die Tatsache, dass beide als Eclipse-Plugins realisiert wurden, so dass die Plattform für die Integration bereits vorhanden ist und lediglich die kommunizierenden Funktionalitäten implementiert werden müssen. Profitieren können davon beide Projekte, da TextGrid ein ausgearbeitetes Nutzer- und Datenmanagement in einer Grid-Infrastruktur bietet, in der im Juli 2011 erschienen Version 1.0 aber keinen Workflow-Editor zur Zusammenstellung von Experimenten über die Daten enthält. Ebenfalls von großem Interesse ist die Inanspruchnahme von wissenschaftlichen *Social Networks*. Dies gilt vor allem für die virtuelle Forschungsum-

gebung *myExperiment*, die für den Austausch von Workflows konzipiert wurde und sich daher eignen würde, Tesla-Experimente in einer elaborierten Umgebung zu publizieren. Die Anfrage hinsichtlich der Aufnahme von Tesla in die unterstützten Workflow-Tools von myExperiment wird mit Veröffentlichung dieser Arbeit gestellt.

Im Laufe der Arbeit am Tesla-System wurde von gleich mehreren Institutionen eine ganze Reihe von Initiativen zur Verbesserung der wissenschaftlichen (Cyber-)Infrastruktur gestartet und mit ansehnlichen Mitteln ausgestattet. Auf europäischer Ebene bildet *ESFRI* die Klammer über derartige Vorhaben, auf deutscher Ebene *D-GRID*. Tesla versteht sich als Beitrag zu diesen Initiativen, auch mit der Zielsetzung, die Textwissenschaft als empirisch arbeitende Wissenschaft zu etablieren. Momentan scheint es nicht vorhersagbar, welche Werkzeuge welcher Institutionen sich letztlich auf dem Markt der verschiedenen Tools und Infrastrukturprojekte durchsetzen werden. Es werden möglicherweise gar nicht die sein, die mit üppigen Fördermitteln von zentralen öffentlichen Einrichtungen protegiert werden, sondern solche, die aus der Innovationskraft von kleinen Startups wie z.B. `Mendeley` hervorgehen. `Mendeley` (`http://www.mendeley.com`, zuletzt aufgerufen am 11.10.2011) ist ein sehr gut ausgearbeitetes Literaturverwaltungsprogramm, das über den Web-Browser, aber auch durch diverse Clients für unterschiedliche Betriebssysteme bedient werden kann. Über die Einrichtung von privaten und öffentlichen Gruppen können Literaturhinweise ausgetauscht und Diskussionsforen eingerichtet werden. So hat sich vor kurzem eine Gruppe zum Thema "Future of Science" konstituiert, in der die Möglichkeiten erörtert werden, den wissenschaftlichen Austausch und damit die Wissenschaft selbst voranzubringen. Über diesen Kommunikationskanal kann man eine Fülle neuer Ansätze und Projekte entdecken und den Eindruck bekommen, die Evolution der Wissenschaft hin zu Open Science habe gerade erst begonnen. Tesla ist unser Beitrag dazu.

Anhang A

Regelhaftigkeit des Chiffrenalphabets aus dem dritten Buch der Trithemischen Polygraphia

Von allen Substitutionstabellen, die Trithemius in seinen Polygraphiae beschreibt, ist die im dritten Buch aufgeführte (im weiteren Verlauf P.III genannte) für diese Arbeit aus zwei Gründen mit Abstand die interessanteste:

1. Ein P.III-verschlüsselter Text weist hinsichtlich der der Wortlängenverteilung und der Abfolge des gleichen Wortes in scheinbar verschiedenen Flexionsformen große Ähnlichkeit mit dem Text des Voynich-Manuskripts auf.
2. Der bereits bei oberflächlicher Betrachtung ins Auge fallende regelmäßige Aufbau von P.III weckt die Hoffnung, einen Text dechiffrieren zu können, der mit einer Tabelle gleicher Art, aber abweichenden Substitutionschiffren, verschlüsselt wurde.

Ein solcher Angriff würde auf der Anordnung der Chiffren in die Zeilen und Spalten beruhen, mithin auf einer Rekonstruktion der als zugrundeliegend gedachten Tabelle. Für diese Rekonstruktion ist eine exakte Beschreibung der Regelmäßigkeiten der Zieldatenstruktur – hier P.III – notwendig, die sich in diesem Anhang findet.

A.1 Analyse der Wörter

Bis auf wenige Ausnahmen, auf die wir in Abschnitt A.5 zurückkommen werden, lassen sich die Wörter von P.III als eine Sequenz dreier Teile analysieren, die man im Rückgriff auf sprachwissenschaftliches Vokabular als Morpheme bezeichnen könnte:[1]

Die Zerlegung der Wörter ist elementar: Jedes Zeichen bis zum letzten Vokal wird dem ersten Morphem zugeschlagen, das zweite Morphem besteht aus jenem letzten Vokal, das fakultative

1 Morpheme sind in der Linguistik definiert als kleinste bedeutungstragende Einheiten. Die Bezeichnung trifft für die Wörter der P.III nur eingeschränkt zu, da hier ganze Wörter im Klartext nur die kleinsten bedeutungsunterscheidenden (und selbst keine Bedeutung tragenden) Einheiten der Schriftsprache, namentlich Grapheme chiffrieren. Dennoch tragen die "Morpheme" der P.III-Wörter insofern Bedeutung, als dass sie Rückschlüsse auf die Position der Wörter in der P.III-Tabelle zulassen.

dritte Morphem wird gebildet von einem potentiell auf den letzten Vokal folgenden Konsonanten. (vgl. Abbildung A.1) Jedes dieser drei Elemente des P.III-Wortmodells unterliegt spezifischen Regelmäßigkeiten, welche den Ansatzpunkt zur Rekonstruktion der P.III ausmachen und mithin der Schlüssel zur Dechiffrierung eines P.III-verschlüsselten Textes sind. Die nächsten drei Abschnitte behandeln die Regelmäßigkeiten dieser drei Morphem-Klassen, der darauf folgende skizziert die Ausnahmen, die Trithemius in P.III zulässt.

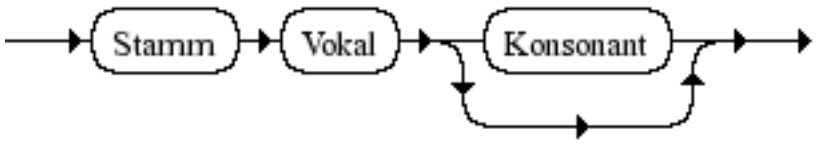


Abbildung A.1: *Morphologie der Wörter aus der P.III-Chiffre*

A.2 Analyse der Stämme

Die Stämme sind zwischen einem und sechs Buchstaben lang, wobei über die Hälfte von ihnen genau drei Zeichen umfassen (Zur Häufigkeitsverteilung der Längen der 130 regelmäßigen Stämme vgl. Abbildung A.2). Sie bestimmen die Spaltenzugehörigkeit eines Wortes. Leider ersann Trithemius nicht 132 verschiedene Stämme, sondern verwendete einige – möglicherweise unabsichtlich – für mehrere Spalten. Insgesamt treten zehn Stämme in zwei Spalten auf, einer (*meda*) sogar in dreien. Diese Tatsache würde kein Problem für den Verschlüsselungsmechanismus bedeuten (gleichwohl für die Tabellenrekonstruktion, aber das mag Trithemius sogar gutheißen mögen, s. Kap. 5.3.1), wenn die Kombinationen aus Stamm, Vokal und Endkonsonant nicht mehrfach bzw. nicht in verschiedenen Zeilen auftreten würden. Unglücklicherweise ist das aber der Fall, was bedeutet, dass man den Text unter Umständen nicht eindeutig entschlüsseln kann, auch wenn die Tabellen, die zu seiner Chiffrierung genutzt wurden, zur Verfügung stehen.[2] Um zwei Beispiele zu geben: *caman* steht in Spalte 65 für x, in Spalte 126 aber für a (vgl. Tabelle A.1, Seite 220, Spalten 1 und 2). *medis* substituiert in Spalte 18 das s, in 47 das n und in 97 das h (vgl. Tabelle A.1, Spalten 3-5). Insgesamt stehen 98 Wörter für zwei verschiedene Buchstaben, 5 sogar für drei. Auf der anderen Seite gibt es damit für 93% der Wörter lediglich einen möglichen Klarbuchstaben; das Maß an Ambiguität ist mithin relativ gering. Die Rekonstruktion indes erschwert sich dramatisch, da wichtige Grundannahmen über den Aufbau der Tabelle nicht eingehalten werden: So ist es für die multipel auftretenden Stämme nicht möglich, alle Formen innerhalb einer einzi-

2 Damit verstößt das Verfahren gegen einen der zentralen Grundsätze der Kryptographie: „[A]ny cipher system, or any method wich claims to follow valid cryptographic procedures, must yield unique solutions.“ (Friedman & Friedman 1959:25)

gen Spalte unterzubringen – die Spaltenzugehörigkeit kann damit nicht mehr eindeutig bestimmt werden. Diese Möglichkeit muss bei einer Rekonstruktion der Tabelle im Auge behalten werden.

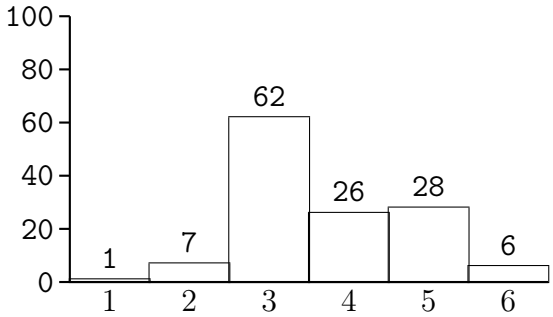


Abbildung A.2: *Verteilung der Häufigkeit (y-Achse) von Stammlängen (x-Achse)*

A.3 Analyse der letzten Vokale

Wo Trithemius in Bezug auf unterschiedliche Stämme bei der Anlage seiner Ersetzungstabelle nicht allzu sorgsam war (oder aber der unbefugten Entschlüsselung mit gewissermaßen unsportlichen Mitteln vorbeugen wollte), verfolgte er in Hinsicht auf den letzten Vokal der Wörter ein klares Konzept: Die Variation des letzten Vokals unterteilt die Zeilen der Tabelle in fünf Blöcke. Jeder Block beginnt mit einem Wort, dessen letzter Vokal das *a* ist, dann folgt ein Wort mit *e*, dann mit *i*, mit *o* und schließlich mit *u* am Ende (das *u* entfällt im letzten Block, da nur 24 Buchstaben verschlüsselt werden). Abweichungen von diesem Muster gibt es keine. Das bedeutet, dass ein Wort mit *a* als letztem Vokal entweder für ein *a*, ein *f*, ein *l*, ein *q* oder ein x[3], nie für irgendeinen anderen Buchstaben steht. Ein Wort mit *e* als letztem Vokal ersetzt demnach immer einen Buchstaben aus der Menge *b,g,m,r,y*; *i* als letzter Vokal lässt auf *c,h,n,s,z* schließen, *o* auf *d,i,o,s,z* und schließlich *u* auf ein Element der Menge *e,k,p,w*. Diese Regel lässt sich in allen Spalten der beiden Tabellen dieses Anhangs beobachten.

A.4 Analyse der Schlusskonsonanten

Die Kenntnis dieser Schlussvokalregel könnte möglicherweise schon genug Information für einen kryptoanalytischen Angriff liefern, da die Menge der möglichen Substitutionen durch Betrachtung des letzten Vokals dramatisch einschränkt werden kann. Damit sind die Regelmäßigkeiten der

3 Zur Erinnerung: Die Abfolge der Buchstaben in den Zeilen der Ersetzungstabelle war, wie in Kapitel 7.3 ausgeführt *a b c d e f g h i k l m n o p q r s t u x y z w*.

Spalte	1	2	3	4	5	6
a	cama	caman	meda	meda	meda	cadas
b	came	camen	mede	mede	mede	cades
c	cami	camin	medi	medi	medi	cadis
d	camo	camon	medo	medo	medo	cados
e	camu	camun	medu	medu	medu	cadus
f	camal	camas	medar	medar	medas	cada
g	camel	cames	meder	meder	medes	cade
h	camil	camis	medir	medir	medis	cadi
i	camol	camos	medor	medor	medos	cado
k	camul	camus	medur	medur	medus	cadu
l	camas	camal	medal	medas	medar	cadan
m	cames	camel	medel	medes	meder	caden
n	camis	camil	medil	medis	medir	cadin
o	camos	camol	medol	medos	medor	cadon
p	camus	camul	medul	medus	medur	cadun
q	camar	camar	medas	medal	medal	cadar
r	camer	camer	medes	medel	medel	cader
s	camir	camir	medis	medil	medil	cadir
t	camor	camor	medos	medol	medol	cador
u	camur	camur	medus	medul	medul	cadur
x	caman	camat	medan	medan	medan	cadal
y	camen	camet	meden	meden	meden	cadel
z	camin	camit	medin	medin	medin	cadil
w	camon	camot	medon	medon	medon	cadol
Originalspalte	65	126	18	47	97	98

Tabelle A.1: *Spalten der P.III mit multipel eingesetzten Stämmen (Spalten 1-5) und fehlendem Endkonsonanten im zweiten Block (Spalte 6)*

Tabelle indes noch nicht erschöpft: Wie aus allen Spalten der Tabelle A.1 unmittelbar ersichtlich, enden die oben beschriebenen, durch den Wechsel ihrer Schlussvokale *a,e,i,o,u* charakterisierten Fünfergruppen, entweder auf diesem ihren Schlussvokal oder immer auf dem gleichen Konsonanten. Diese Information resultiert in einer Fülle von Abhängigkeiten zwischen den Wortchiffren (die in A.2 beschriebenen ambigen Stämme seien hier aus unserer Betrachtung ausgeschlossen): Wenn ein Wort *STAMM-a-l* für *a* steht, so substituiert das Wort *STAMM-e-l* das *b*, *STAMM-i-l* das *c*, *STAMM-o-l* das *d* und *STAMM-u-l* ersetzt den Buchstaben *e*. Auch von dieser Regel gibt es in der gesamten Tabelle keine Ausnahme, Wörter innerhalb der Fünferblöcke unterscheiden sich also immer nur durch ihren letzten Vokal. Leider lässt sich keine eindeutige Zuordnung der Schlusskonsonanten zu den einzelnen Fünferblöcken leisten. Wie aus Tabelle A.1 ersicht-

lich, kann der Schlusskonsonant *n* sowohl für den ersten Block (Tabellenspalte 1), für Block 3 (Spalte 6) wie auch für Block 5 stehen (Spalten 1,3-5). Analysiert man sämtliche Spalten der P.III, so scheint jeder Konsonant für jeden beliebigen Block stehen zu können. Diese Unregelmäßigkeit dürfte fraglos auf Trithemius' Streben nach einer sicheren Chiffre zurückzuführen sein. Wären die Schlusskonsonanten nur bestimmten Blöcken zugeordnet, ergäbe sich in Kombination mit dem vorangehenden Vokal ein Suffix, das eindeutig einem Buchstaben zugeordnet werden könnte, so dass das System einer monoalphabetischen Verschlüsselung (zwar mit eventuellen Suffix-Homophonen und Blendzeichen am Wortanfang) mit all ihren in Kapitel 5.2 aufgeführten Nachteilen entsprechen würde.

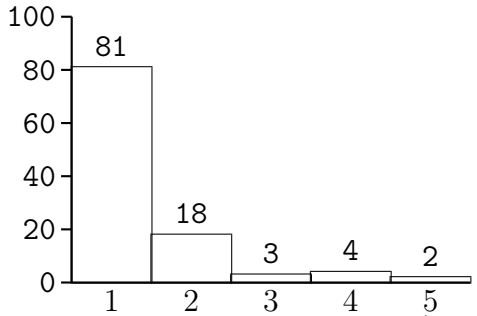


Abbildung A.3: *Anzahl der fehlenden Schlusskonsonanten (y-Achse) über die fünf Blöcke der Zeilen aus P.III (x-Achse).*

Trotz der offensichtlichen unregelmäßigen Verteilung der Schlusskonsonanten gibt es in ihr eine Auffälligkeit, die zwar eher statistischen Charakter hat, für die Kryptoanalyse aber dennoch nützlich sein könnte: Schreibt man die Wörter der einzelnen Zeilen hintereinander, so fällt ins Auge, dass die Zeilen von oben nach unten länger werden. Der Buchstabe *a* wird bspw. im Durchschnitt von kürzeren Wörtern codiert als der Buchstabe *x*. Dieses Phänomen begründet sich v.a. durch die Fakultativität des Schlusskonsonanten: In 94 der 130 regelmäßigen Spalten gibt es jeweils einen Block, dessen Wörter alle auf dem letzten Vokal enden. Bei den Wörtern der anderen vier Blöcke dieser Spalten folgen auf den letzten Vokal noch Konsonanten, wie oben beschrieben, je ein signifikanter für jeden Block. Damit sind diese Blöcke natürlich durchschnittlich länger als der schlusskonsonantenlose Block. Dadurch, dass letztere v.a. im ersten Block zu finden sind (vgl. Abbildung A.3), ergeben sich in der Summe die oben erwähnten summierten Wortlängenunterschiede zwischen den einzelnen Blöcken. Wie an der Abbildung ersichtlich, ist es nicht sicher, dass ein Wort, das auf *-o* endet, tatsächlich ein *d* repräsentiert, es ist aber doch wahrscheinlicher, als dass es ein *h* substituiert.[4]

4 Tatsächlich muss natürlich noch die Vorkommenswahrscheinlichkeit eines Buchstaben in die Berech-

A.5 Die Ausnahmen

Schon die einzelnen Morpheme der P.III-Wörter weisen diverse Auffälligkeiten auf. Trithemius lässt aber auch Ausnahmen zu, die zu Abweichungen vom in Abb. A.1 dargestellten Schema führen. Diese Ausnahmen betreffen in allen Fällen nur einzelne oder eine zumindest sehr begrenzte Zahl von Spalten und unterscheiden sich im Grad ihrer Abweichungen vom Grundmuster. Alle Ausnahmen erschweren die Aufgabe der Tabellenrekonstruktion. Die nachfolgende Aufstellung versucht, die Phänomene aufsteigend nach der Stärke ihrer Abweichung zu ordnen.

A.5.1 Konsonantenverdoppelung

Ist der Endkonsonant ein *f*, so wird dieses verdoppelt. Dieses Phänomen findet sich im Block 2 der Spalte 7 und Block 5 der Spalte 15 (vgl. Tabelle A.2 Spalte 1). Diese Abweichung lässt sich recht einfach in den Griff bekommen, da sie lediglich und v.a. konsequent bei genau einem Schlusskonsonanten – dem *f* – auftritt.

A.5.2 Zwei Vokale am Wortende

Statt eines Schlusskonsonaten hängt Trithemius bisweilen ein *i* an den signifikanten Vokal, der damit nicht mehr der letzte, sondern der vorletzte ist. Dies ist meistens im fünften Block der Fall (Spalten 103, 106, 109, 111 und 132), zum Ende der Tabelle hin im vierten (Spalten 128-131). Bei zwei aufeinanderfolgenden *i* wird das zweite als *j* geschrieben (vgl. Tabelle A.2 Spalte 2). Wichtig ist hier, dass der Stamm vor dem vorletzten Vokal endet. Da nur die betreffenden Blöcke auf zwei Vokalen enden, lässt sich auch diese Ausnahme vergleichsweise einfach behandeln.

A.5.3 Stammverlängerung

In einigen Fällen werden die Stämme innerhalb einer Spalte dergestalt variiert, dass sie in einem Block um ein Vokal+Konsonant-Suffix verlängert werden, an das wiederum der Schlussvokal und eventuell der Schlusskonsonant angehängt werden (vgl. *baschay*, Tabelle A.2, Spalte 3, v.a. Block 4). Dies ist der Fall in Spalte 109, Block 5, 124/4 und 131/2. In einem einzigen Fall wird der Stamm in gleich vier Blöcken verlängert (aus anderer Sicht: In einem verkürzt), und zwar bei der *af*-Reihe in Spalte 89 (vgl. Tabelle A.2 Spalte 4, Blöcke 2-5). Diese Ausnahme ist für die Rekonstruktion knifflig, da das Prinzip vom eindeutigen Stamm pro Spalte unterlaufen wird.

nung mit einbezogen werden, der Einfachheit halber gehen wir für dieses Beispiel davon aus, dass *d* und *h* gleich häufig im Klartext auftreten).

A.5.4 Reduplikation

In der vierten der 132 Spalten (*cadalan*) findet sich eine Abweichung im Schema, die an das Prinzip der Reduplikation aus der lateinischen Grammatik erinnert, denn hier wird nicht nur der letzte Vokal variiert, sondern der vorletzte im gleichen Maße: Statt der erwarteten Form *cadalen* findet sich in der zweiten Zeile *cadelen*, in der dritten *cadilin* etc. (vgl. Tabelle A.2 Spalte 5). Auch hier gilt das Prinzip des eindeutigen Stammes pro Spalte nicht, so dass auch durch diese – offenbar durch ein Phänomen der natürlichen Spachen beeinflusste – Ausnahme die Rekonstruktion erschwert wird.

Spalte	1	2	3	4	5	6
a	maſtra	faſan	baſchay	afar	cadalan	darach
b	maſtre	faſen	baſchey	afer	cadelen	darech
c	maſtri	faſin	baſchiy	afir	cadilin	darich
d	maſtro	faſon	baſchoy	afor	cadolon	daroch
e	maſtru	faſun	baſchuy	afur	cadulun	daruch
f	maſtran	faſal	baſchar	afara	cadalas	derach
g	maſtren	faſel	baſcher	afare	cadeles	derech
h	maſtrin	faſil	baſchir	afari	cadilis	derich
i	maſtron	faſol	baſchor	afaro	cadolos	deroch
k	maſtrun	faſul	baſchur	afaru	cadulus	deruch
l	maſtral	faſar	baſchal	afaral	cadalap	dirach
m	maſtrel	faſer	baſchel	afarel	cadelep	direch
n	maſtril	faſir	baſchil	afaril	cadilip	dirich
o	maſtrol	faſor	baſchol	afarol	cadolop	diroch
p	maſtrul	faſur	baſchul	afarul	cadulup	diruch
q	maſtras	faſat	baſchata	afaras	cadalar	dorach
r	maſtres	faſet	baſchate	afares	cadeler	dorech
s	maſtris	faſit	baſchati	afaris	cadilir	dorich
t	maſtros	faſot	baſchato	afaros	cadolor	doroch
u	maſtrus	faſut	baſchatu	afarus	cadulur	doruch
x	maſtraff	faſai	baſchas	afarat	cadalaz	durach
y	maſtreff	faſei	baſches	afaret	cadelez	durech
z	maſtriff	faſii	baſchis	afarit	cadiliz	durich
w	maſtroff	faſoi	baſchos	afarot	cadoloz	durich
Originalspalte	15	130	124	89	4	41
Block	5	5	4	1	Alle	Alle

Tabelle A.2: *Spalten der P.III mit Beispielen für unregelmäßige Formenbildung. Angabe der bemerkenswerten Blöcke in der letzten Zeile.*

A.5.5 Ablaut

Die (aus linguistischer Sicht) interessanteste Variation des Schemas findet sich in Spalte 41 (*dar-ach*): Sämtliche Formen enden auf dem Konsonantenpaar *-ch*, die Unterscheidung der Fünfergruppen muss also auf anderem Wege erfolgen. An der Funktion des Schlussvokals ändert sich nichts, weiterhin weist jede Fünfergruppe die Abfolge *a,e,i,o,u* auf. Aber in dieser Spalte kombiniert Trithemius den Endvokal mit dem ersten Vokal, er lautet also gleichsam den Stamm ab, so dass sich 5*5, also 25 verschiedene Formen ergeben, von denen er 24 für seine Tabelle nutzt (die Kombination *u-u* ist überflüssig, vgl. Tabelle 2 Spalte 6). Obschon auch durch dieses Mittel ein einfacher Spaltenentwurf möglich gewesen wäre, nutzt Trithemius es, wie die Reduplikation, nur ein einziges Mal, als wolle er sein Wissen um diese Möglichkeit der regelmäßigen Tabellenanlage aufzeigen. Wahrscheinlich war ihm diese Methode schlichtweg zu unsicher, als dass er sie öfter einsetzen wollte, da sich aus der Kombination der beiden Vokale eindeutig der Klarbuchstabe ergibt. Die Vokal-Konsonantenmethode, wobei die Konsonaten keine festen Blöcke codieren (vgl. oben A.4), ist das eindeutig sichere System, auch wenn es die Arbeit der Kryptoanalytiker, die lediglich einen Text, aber keinen Schlüssel in Form von Tabellen haben, erschwert.

Anhang B

Entropie

Der Begriff Entropie wurde von Shannon (1948) auf die Informationstheorie übertragen. Ursprünglich stammt er aus der Thermodynamik, wo Entropie – etwas vereinfacht gesagt – das Maß an Ordnung oder Unordnung (je nachdem, was man als Ordnung definiert) bezeichnet. In der Informationstheorie steht die Entropie in unmittelbarem Zusammenhang mit dem Informationsgehalt. Der Informationsgehalt b einer beliebigen Einheit eines Zeichensystems I, ausgedrückt in Bits, ist der negative binäre Logarithmus der Auftrittswahrscheinlichkeit $p(I)$ dieses Zeichens:

$$b(I) = -log_2(p(I)) \tag{B.1}$$

Die Entropie h_1 ist nun ein Maß für den mittleren Informationsgehalt H eines Zeichensystems mit n unterschiedlichen Ereignissen I. Sie wird über die Summe der Produkte aller Wahrscheinlichkeiten von I mit dem Informationsgehalt aller I ermittelt:

$$h_1 = H(I) = \sum_{I=1}^{n} p(I) * b(I) \tag{B.2}$$

Nach Einsetzen der Formel B.1 in B.2 ergibt sich dann

$$h_1 = H(I) = -\sum_{I=1}^{n} p(I) * log_2(p(I)) \tag{B.3}$$

B.1 Interpretation der Entropie

Setzen wir als Zeichensystem einen Text, so liegt der Informationsgehalt eines Zeichens innerhalb dieses Textes umso höher, je seltener das Zeichen im Text vorkommt. Wenn der Text ausschließlich aus einem Zeichen besteht, ist der Informationsgehalt dieses Zeichens (und des gesamten Textes) 0. Die Entropie als mittlerer Informationsgehalt eines Textes wäre in diesem Fall 0. Was aber ist der höchste mittlere Informationsgehalt, also die maximale Entropie H_{max} eines Textes? H_{max}, auch als h_0 bezeichnet, wird erreicht, wenn alle Zeichen gleich oft vorkommen. Der Wert

hängt von der Anzahl der unterschiedlichen Zeichen ab, da

$$h_0 = H_{max} = -\sum_{I=1}^{n} \frac{1}{N} * log_2 \frac{1}{N} = log_2 N \tag{B.4}$$

Für Texte mit 30 verschiedenen Einheiten (wie dem deutschen Alphabet inklusive Umlauten und ß) liegt die maximale Entropie bei 4,91, in Zeichensystemen mit nur 10 Einheiten (z.B. Ziffernfolgen) liegt sie bei 3,32. Die maximale Entropie kann dazu genutzt werden, um unterschiedliche Texte mit einem unterschiedlich umfangreichen Zeicheninventar miteinander zu vergleichen, indem man die ermittelten Entropien der Texte durch die maximalen Entropiewerte für diese Texte teilt. Der damit ermittelte Entropiewert bleibt immer kleiner als 1.

Die Entropie eines Zeichensystems liegt demnach immer zwischen 0 (wenn der Text nur aus einem Zeichen besteht) und H_{max}(Wenn alle Zeichen gleich oft vorkommen). Die Entropiewerte sind also, wenn man nur einzelne Zeichen betrachtet, lediglich ein Maß für die Gleichverteilung von Zeichen eines Textes. Wenn nun aber nicht nur Einzelzeichen, sondern auch Zeichenkombinationen als Zeichen zugelassen werden, kann die Entropie als ein Maß für die Repetitivität von Texten angesehen werden, d.h. als Maß für die Wiederkehr von Mustern in Texten. Zur Ermittlung der Entropiewerte für Zeichenkombinationen muss die Verbundentropie, auch Blockentropie genannt, errechnet werden.

B.2 Verbund- oder Blockentropie

Die Blockentropie H_k, wobei k die Anzahl der Zeichen von Blöcken angibt, basiert auf der Verbundwahrscheinlichkeit von Zeichen $p(I, J)$. Diese ergibt sich aus der Beziehung:

$$p(I, J) = \frac{p(J|I)}{(p(I))} \tag{B.5}$$

Dabei ist $p(J|I)$ die sogenannte *Bedingte Wahrscheinlichkeit*, also die Wahrscheinlichkeit für das Auftreten von I unter der Bedingung, dass J gegeben ist. $p(\text{“}a\text{“}|\text{“}\_\text{“})$, verbalisiert “Wahrscheinlichkeit für das Zeichen a unter der Bedingung, dass ein Leerzeichen vorangeht” wäre mithin die Wahrscheinlichkeit, dass ein Wort mit “a” beginnt.

Der Informationsgehalt eines Blocks der Länge 2 mit J als erstem und J als zweitem Element ist nun folgendermaßen definiert:

$$b(I, J) = -log(p(I, J)) \tag{B.6}$$

Daraus ergibt sich für die Verbundentropie von Zweierblöcken h_2, in diesem Fall auch die bedingte

Entropie oder Entropie zweiter Ordnung:

$$h_2 = \sum_{I=1}^{n} p(I,J) * b(I,J) = -\sum_{I=1}^{n}\sum_{J=1}^{n} p(x_I) * p(y_J|x_I) * log_2(p(y_J|x_I)) \tag{B.7}$$

Entropien höherer Ordnung ermittelt man entsprechend, wobei unschwer zu erkennen ist, dass schon die Berechnung von h_3 sehr kostenintensiv ist:

$$h_3 = \sum_{I} p(I) \sum_{J} p(I,J) \sum_{K} P(I,J,K) - log_2(P(I,J,K)) \tag{B.8}$$

Anhang C

Komponentendokumentation

Im Folgenden findet sich ein Überblick über die im Rahmen dieser Arbeit beschriebenen und eingesetzten Tesla-Komponenten. Nach einer Kurzcharakterisierung werden die wichtigsten Informationen zu den Schnittstellen der Komponenten in tabellarischer Form zusammengefasst. Dabei werden die nachstehenden Fragen in kompakter Form beantwortet:

1. Welche Rolle übernimmt die Komponente, d.h. was produziert sie?
2. Welche Rollen werden als Input der Komponente benötigt, d.h. was konsumiert sie?
3. Welche Datenbank wird genutzt?
4. Wer hat die Komponente entwickelt?
5. Welche Möglichkeiten der Konfiguration weist die Komponente auf?
6. Welche Funktion erfüllt die Komponente?

Aufgeführt werden zunächst die in den Kapiteln 3.1.3 und 6.1.4 erwähnten Reader-Komponenten (C.1), anschließend die in 6.1.4 und 6.2.4 genutzten Komponenten (C.2). Auf eine sequenzielle Aufführung der beteiligten Rollen wurde verzichet, da Rollen am besten in einer hierarchischen Aufstellung des *Tesla Role System* darzustellen sind, wie dies in der Online-Dokumentation unter `http://tesla.spinfo.uni-koeln.de/group__trs.html` geschieht.

C.1 Reader

Die Aufstellung der Reader beginnt mit dem unspezifischsten Tesla-Reader, dem `TextReader`. Der danach vorgestellte `XMLReader` ist auf XML-Dateien spezialisiert, ohne jedoch die XML-Auszeichnungen zu interpretieren. Letzteres vermögen die anschließend aufgeführten `SpinfoCorpusReader` und `BNCReader` für die entsprechenden XML-Formate. Zum Schluss werden zwei weiterere spezialisierte Reader aufgeführt, zum Einen der `VoynichInterlinearArchiveFileReader`, der Dateien im VIAF (siehe 4.2.2) interpretieren kann, zum anderen der `BibleReader` der auf das flache Text-Format von Bibel-Dateien spezialisiert ist und dabei einzelne Verse als Paragraphen labelt.

Weitere bereits in Tesla implementierte, aber im Rahmen dieser Arbeit nicht genutzte Reader

sind der `TigerCorpusReader` für das Tiger-Format[1], der `ReutersXMLReader` für das Reuters-Format[2], der `FASTAReader` für genetische Daten im FASTA-Format[3], der `MIDIReader` für das MIDI-Format[4], der `BrownTEIReader` für das Brown-Korpus[5]-Format sowie der `TIKAReader` für gängige Textdateiformate (`pdf, doc, odf` etc.)[6].

Reader besitzen unterschiedliche Spezialisierungsgrade (je höher der Wert, desto spezialisierter ist der Reader, d.h. desto eingeschränkter ist das Format, was er konsumieren kann, vgl. 3.1.3), damit Tesla entscheiden kann, welcher Reader für die Prozessierung von Texten ausgewählt werden soll, wenn meherere Reader das Format unterstützen. Spezialisierte Reader (d.h. Reader mit höherem Spezialisierungsgrad) werden generellen Readern vorgezogen. Der Grad der Spezialisierung ist für jeden Reader in der Tabellenspalte 'Funktion' aufgeführt.

C.1.1 Text Reader

Der `TextReader` ist in Tesla der Default-Reader für alle zeichenbasierten Daten. Er wird immer dann ausgewählt, wenn kein anderer Reader den Inhalt der Korpusdatei unterstützt. Liefert lediglich alle Bytes der Datei, kann keine Meta-Informationen, Formatierungen oder sonstige Annotationen erkennen.

Text Reader	
Produziert	–
Konsumiert	Signale
Datenbank	–
Autor	Jürgen Hermes
Konfiguration	–
Funktion	Liest den Inhalt (Bytes) sämtlicher Dateien auf einen InputStream. Spezialisierungsgrad 20

1 Zum Tiger-Korpus, einer syntaktisch ausgezeichneten Baumbank vgl. `http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/doc/html/TigerXML.html` (zuletzt abgerufen am 11.10.2011).

2 Zum Reuters-Korpus vgl. `http://trec.nist.gov/data/reuters/reuters.html` (zuletzt abgerufen am 11.10.2011).

3 Zum FASTA-Format für die Primärstrukturen von DNA, RNA und Proteinen vgl. `http://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml` (zuletzt abgerufen am 11.10.2011).

4 Zur Spezifikation des MIDI-Formats für musikalische Steuerinformationen vgl. `http://home.snafu.de/sicpaul/midi/midi0a.htm` (zuletzt abgerufen am 11.10.2011).

5 Zum Brown-Korpus vgl. `http://icame.uib.no/brown/bcm.html` (zuletzt abgerufen am 11.10.2011).

6 Tika ist ein Projekt der Apache Software Foundation, das ein Toolkit für die Extraktion von Metadaten und strukturiertem Text verschiedener Dokumentformate durch bestehende Parser-Bibliotheken bereitstellt, vgl. `http://tika.apache.org/`, zuletzt abgerufen am 11.10.2011.

C.1.2 XML Reader

Default-Reader für XML-Dateien. Behandelt keine Auszeichnungen, dafür sind spezialisiertere Reader (SpinfoCorpusReader, BNCReader, s.u.) zuständig.

XML Reader	
Produziert	–
Konsumiert	Signale
Datenbank	–
Autor	Stephan Schwiebert
Konfiguration	–
Funktion	Liest den Inhalt (Content) von XML-Dateien (ohne Auszeichnungen) auf einen InputStream. Spezialisierungsgrad 50

C.1.3 Spinfo Corpus Reader

Dieser Reader wurde für ein von der Sprachlichen Informationsverarbeitung zu Testzwecken erstelltes Zeitungskorpus entworfen. Dieses liegt in einem einfach gehaltenen XML-Dialekt vor, der lediglich Texte und Paragraphen unterteilt und zusätzlich Meta-Informationen nach dem Dublin-Core-Standard für die Texte bereithält. Sowohl die Text- und Paragraphgrenzen, als auch die Dublin-Core-Metadaten werden vom Reader als Annotationen gespeichert.

Spinfo Corpus Reader	
Produziert	DublinCoreMetadataGenerator, ParagraphDetector
Konsumiert	Signale
Datenbank	Hibernate
Autor	Jürgen Hermes
Konfiguration	–
Funktion	Liest Texte im Spinfo-XML-Format, liefert neben dem Text Annotationen zu Paragraphen und Dublin-Core-konforme Metadaten. Spezialisierungsgrad 80

C.1.4 BNC Reader

Der BNC-Reader ist spezialisiert auf Texte aus dem BNC-Korpus. Er liefert neben deren Content auch die in diesen Dateien codierten Informationen (Wortgrenzen, Satzgrenzen, PoS-Tags, Lemmata). Dabei produziert er die zu diesen Auszeichnungen korrespondierenden Tesla-Rollen.

BNC Reader	
Produziert	Tokenizer, SentenceDetector, ParagraphDetector, Lemmatizer, PoSTagger
Konsumiert	Signale
Datenbank	TunguskaDB
Autor	Sebastian Rose
Konfiguration	–
Funktion	Liefert Inhalt (Content) von Texten aus dem BNC-Korpus. Konvertiert BNC-Auszeichnungen in Tesla-Rollen. Spezialisierungsgrad 100

C.1.5 Bible Reader

Dieser Reader wurde speziell für das typische flache Bibel-Textformat (einzelnen Versen sind Nummern vorangestellt) entworfen. Verse werden vom Reader als Paragraph-Annotationen gespeichert.

Bible Reader	
Produziert	ParagraphDetector
Konsumiert	Signale
Datenbank	Hibernate
Autor	Jürgen Hermes
Konfiguration	–
Funktion	Liest Texte im flachen Bibel-Format, liefert neben dem Text auch Annotationen zu einzelnen Versen. Spezialisierungsgrad 40

C.1.6 Voynich Interlinear Archive File Reader

Dieser Reader wurde eigens für das von Stolfi editierte Voynich Interlinar Archive File (VIAF) entworfen, in dem die geläufigsten Transkriptionen des Voynich-Manuskripts gelistet sind. In der Konfiguration können spezifische Transkriptionen in Verbindung mit einzelnen Sektionen und/oder Abschnitten spezifizierter Currier-Sprachen ausgewählt werden. Der Text kann entweder mit Füllzeichen und In-Line-Kommentaren oder ohne diese geliefert werden, darüber hinaus kann man spezifizieren, in welchem Transkriptionsalphabet man die Zeichen codiert haben möchte (EVA oder Currier). Die für die einzelnen Seiten und Zeilen des VM angegebenen Meta-Informationen werden zusätzlich als Annotationen abgelegt.

Voynich Interlinear Archive File Reader		
Produziert	Classifier	
Konsumiert	Signale	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	All Transcriptions	Spezifiziert, ob alle Transkriptionen geliefert werden sollen
	Transcription Choice	Spezifiziert die zu liefernde Transkription
	Section Choice	Spezifiziert die zu liefernde VM-Sektion
	Currier Hand Choice	Spezifiziert die zu liefernde Currier-Sprache
	Currier Alphabet	Spezifiziert das Alphabet, in dem der Text geliefert wird (Currier oder EVA)
	Delete Fillers	Löschen der Füll-Tokens aus dem VM-Text
	Delete Comments	Löschen der Kommentare aus dem VM-Text
Funktion	Liefert den Voynich-MS Text aus dem von Stolfi editierten VIAF. Spezialisierungsgrad 100	

C.2 Komponenten

Die hier aufgeführten Komponenten sind diejenigen, die in den Experimenten in Kapitel 6 eingesetzt werden. Auf die nähere Beschreibung aller bisher implementierten Tesla-Komponenten wurde aus Übersichtsgründen verzichtet, hier sei auf die ausführliche Online-Dokumentation unter `http://tesla.spinfo.uni-koeln.de/group__components.html` verwiesen.

C.2.1 SPre

SPre ist eine der ältesten Tesla-Komponenten, der Präprozessor wurde bereits vor der Entwicklung des Tesla-Systems als eigenständiges Programm entwickelt, das selbst einer Komponentensystem-Architektur folgt (vgl. Kap. 3.1.4) und mit Wrappern für die damals schon bestehenden Komponentensysteme (Gate, UIMA) versehen wurde.

Das zentrale Charakteristikum von SPre ist die konsequente Konfigurierbarkeit: Auf der einen Seite kann festgelegt werden, auf welchen Ebenen (Layer) der Input tokenisiert werden soll. Die Zeichenebene (CharacterLayer) ist dabei obligatorisch, weil sie die Basis für alle weiteren Ebenen ist, die letztlich aus Sequenzen von Einheiten der Zeichenebene bestehen. Oberhalb des CharacterLayers befindet sich normalerweise der WordLayer, auf dem Wörter, Zahlen, aber auch die Satzzeichen als solche erkannt werden. Darüber können sich Sentence- und ParagraphLayer

anschließen, möglich sind aber auch Layer, die etwa Passagen wörtlicher Rede ermitteln oder unterschiedliche Formatierungen des Textes auswerten.

Auf jedem dieser frei wählbaren Layer können nun Sequenzen als Tokens dieses Layers gelabelt werden. Auch die Regeln für die Ermittlung der Tokens sind Teil der SPre-Konfiguration: Jeder Layer benötigt einen Konfigurations-Text, mit dem der zugehörige Parser den Input analysiert. Diese Konfiguration besteht momentan noch aus einem XML-Dialekt und wird in einer zukünftigen Version von SPre mit Hilfe eines XML-Editors der Tesla-Konfigurationsschnittstelle (vgl. Kap. 3.2.3) gesteuert werden können.

SPre Component		
Produziert	SentenceDetector, Tokenizer	
Konsumiert	ReaderSignals	
Datenbank	TunguskaDB	
Autor	Jürgen Hermes, Christoph Benden	
Konfiguration	Parser Configurations	Komplexe Konfiguration des Tokenizers, in dem die unterschiedlichen Layer und Tokens definiert werden, dazu deskriptive Definition der Tokenisierungsalgorithmen.
	Abbreviation List	Liste der zu verwendenden Abkürzungen
Funktion	Tokenisiert den Eingabetext auf den in der Konfiguration festgelegten Ebenen (Layer, bspw. Character-, Word-, SentenceLayer). Ist durch die konsequente Konfigurierbarkeit für jede algorithmisch ausdrückbare Tokenisierung einsetzbar.	

C.2.2 Cryptographic Substitution Component

Diese Komponente enthält eine Reihe von kryptologischen Methoden, mit denen Texte ver- bzw. entschlüsselt werden können. Die Auswahl der Verfahren basiert auf den Erfordernissen des in dieser Arbeit gestellten Themas, die Komponente kann aber jederzeit mit zusätzlichen Methoden erweitert werden. Bisher wurden implementiert:

- Atbash, Cäsar (kryptographisch, monoalphabetisch, monographisch, monopartit)
- Polybus (kryptographisch, monoalphabetisch, monographisch, bipartit)
- Playfair (kryptographisch, monoalphabetisch, bigraphisch, bipartit)
- Vigenère, Trithemius' Polygraphia V (kryptographisch, polyalphabetisch, monographisch, monopartit)
- Trithemius' Polygraphia I, II und III (steganographisch, maskiert)

- Trithemius' Steganographia I und Polygraphia IV (steganographisch, getarnt, Würfel)
- Gematria (Summenberechnung)

Soweit es sich nicht – wie bei der Gematria-Umrechnung – um Einwegverfahren handelt, sind jeweils beide Richtungen der Substituton (also Ver- und Entschlüsselung) implementiert.

Für die Konfiguration der Komponente steht ein umfangreiches `TemplateSet` (vgl. Kap. 3.2.3) zur Verfügung, das für die jeweiligen Verfahren passende Default-Werte (etwa Schlüssel im benötigten Format setzt – z.B. eine natürliche Zahl für die Methode von Caesar, eine Zeichenkette für die von Vigenère, eine Folge boolscher Werte für die in der Steganographia I beschriebene Methode.

Cryptographic Substitution Component		
Produziert	TokenSubstitutor	
Konsumiert	Tokenizer	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Substitution Method	Auswahl des Substitutionsverfahrens
	Direction	Festlegen der Anwendungsrichtung (Verschlüsselung oder Entschlüsselung)
	Key	Setzen evtl. benötigter Schlüssel (unterschiedliche Formate wie `int`, `String` und `boolean[]` sind zugelassen), wird vom ausgewählten Verfahren interpretiert
	Alphabet	Festlegen des Klartextalphabets
	Substitutes	Festlegen eines vom Klartextalphabet abweichenden Geheimtextalphabets
	Replacements	Ersetzungseinheiten im Klartext (bspw. ä zu ae)
	Whitespace Deletion	Festlegen der Ersetzungstextformatierung
Funktion	Ver- bzw. entschlüsselt die übergebenen Token-Einheiten mit dem in der Konfiguration festgelegten Verfahren.	

C.2.3 Corpus Statistics Component

Diese Komponente berechnet eine Reihe von statistischen Werten der tokenisierten Texte des Inputs. Da die Berechnungen bisweilen sehr zeitaufwendig sein können, kann man in der Konfiguration festlegen, welche Werte berechnet werden sollen. Darüber hinaus ist wählbar, ob die statistischen Werte zu einzelnen Dokumenten oder zu ganzen Dokument-Selektionen ermittelt werden sollen.

Als Output werden Objekte vom Typ `CorpusStatsVector` generiert, die aus Attribut-Wert-Paaren bestehen, die aber auch – zur Weiterverarbeitung in Vektor-konsumierenden Komponenten wie z. B. dem Clusterer – auf einem Vektor vorgehalten werden.

Corpus Statistics Component		
Produziert	CorpusStatisticsVectorGenerator	
Konsumiert	Tokenizer, ReaderSignals	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Character Entropy	Berechnung Zeichenentropie
	Word Entropy	Berechnung Wortentropie
	Joint Character Entropy	Berechnung Verbundentropie Zeichen
	Joint Word Entropy	Berechnung Verbundentropie Wörter
	Conditional Character Entropy	Berechnung bedingte Entropie Zeichen
	Conditional Word Entropy	Berechnung bedingte Entropie Wörter
	nChar Entropy	Berechnung der Entropie für einzelne Positionen in Wörtern
	Word Length Statistics	Berechnung Wortlängenstatistik
	Type Token Ratio	Berechnung Type/Token-Verhältnis
	Zipfian Constant	Berechnung Zipf-Konstante
	Label Based Statistics	Auswertung des Token-Strings vs. Auswertung des Token-Labels
	To Lower Case	Großbuchstaben beibehalten vs. zu Kleinbuchstaben umwandeln
	Statistics Per Document	Statistik über die gesamte Selection vs. Statistik für einzelne Dokumente
Funktion	Berechnet die in der Konfiguration ausgewählten Statistiken zu den tokenisierten Texten des Inputs.	

C.2.4 Coincidence Statistics Component

Diese Komponente berechnet Koinzidenzwerte von Texten, welche in der Kryptographie bzw. (genauer) in der Kryptoanalyse eine Rolle spielen. Dabei kann ausgewählt werden, ob die Berechnung textintern erfolgen soll (dabei wird jeder Text in zwei gleich lange Zeichenketten zerlegt, deren Koinzidenzwerte errechnet werden) oder ob die Koinzidenzwerte für alle möglichen Textpaare ermittelt werden sollen. In letzterem Fall sind Kappa und Chi die Mittelwerte aller Vergleichsoperationen. Die Objekte des Outputs (`CoincidenceStatsVector`) enthalten die berechneten Kennwerte sowohl als Attribut-Wert-Paare, wie auch aggregiert auf einem Vektor.

Coincidence Statistics Component		
Produziert	CoincidenceStatisticsVectorGenerator	
Konsumiert	Tokenizer, ReaderSignals	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Label Based Statistics	Auswertung des Token-Strings vs. Auswertung des Token-Labels
	To Lower Case	Großbuchstaben beibehalten vs. zu Kleinbuchstaben umwandeln
	Statistics Per Document	Koinzidenzwerte werden entweder pro Text ermittelt (erste vs. zweite Häfte) vs. im Vergleich aller Texte miteinander
Funktion	Berechnet Koinzidenzwerte (Kappa, Chi, Phi und Psi) der Input-Texte.	

C.2.5 Random Walk Component

Diese Komponente simuliert einen Random Walk über den spezifizierten Texten. Über eine Konfigurationsschnittstelle können die vom Walk zu beachtenden Zeichen festgelegt werden, dazu kann die Laufzeit verkürzt werden, indem das größte zu berechnende Intervall angegeben wird (die Laufzeit ist quadratisch zu dieser Intervallgröße). Ausgabe der Komponente sind ein Vektor über die Intervall-Standardabweichungen und ein Vektor über die Steigung der auf eine log-log-Matrix aufgetragenen Intervall-Standardabweichungen (in der Literatur als alpha bezeichnet).

Random Walk Component		
Produziert	DoubleVectorGenerator (2)	
Konsumiert	Tokenizer, ReaderSignals	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Alphabet	Alphabet der zugelassenen Walk-Zeichen
	Interval Calculation Bound	Größtes auszuwertendes Intervall
	Label Based Statistics	Auswertung des Token-Strings vs. Auswertung des Token-Labels
	To Lower Case	Großbuchstaben beibehalten vs. zu Kleinbuchstaben umwandeln
	Statistics Per Document	Statistik über die gesamte Selection vs. Statistik für einzelne Dokumente
Funktion	Simuliert einen Random Walk, berechnet die Intervall-Standardabweichungen sowie die Steigung der Intervall-Abweichungskurve.	

C.2.6 Repeated Words Detector

Diese Komponente zählt die Vorkommen von gleichen oder ähnlichen Wörtern (Minimalpaaren), die direkt aufeinander folgen.

Repeated Words Detector		
Produziert	Counter	
Konsumiert	Tokenizer, ReaderSignals	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Label Based Statistics	Auswertung des Token-Strings vs. Auswertung des Token-Labels
	To Lower Case	Großbuchstaben beibehalten vs. zu Kleinbuchstaben umwandeln
	Statistics Per Document	Statistik über die gesamte Selection vs. Statistik für einzelne Dokumente
Funktion	Zählt Wortzwillinge und Wortdrillinge (gleiche, direkt adjazente Wörter sowie Minimalpaarzwillinge und -drillinge (adjazente Wörter mit Levensthein-Distanz 1)	

C.2.7 Graphemizer

Im Graphemizer ist die in Kapitel 6.2.1 beschriebene Minimalpaarmethode zur Ermittlung von Graphemen und ihren distributionellen Eigenschaften implementiert. Die Komponente füllt zwei Rollen aus, da zwei inhaltsgleiche, aber in der Implementierung unterschiedliche Datenobjekte erzeugt werden: Einerseits werden die distributionellen Daten zu Graphemen gegliedert in einem Graphem-Objekt gespeichert, andererseits werden sie als relative Werte in einen Vektor repräsentiert, so dass sie für Vektor-konsumierende Komponenten (bspw. einen Clusterer) verarbeitbar sind. Die Minimalpaare werden über Types ermittelt, deshalb benötigt der Graphemizer als Input nur jeweils ein Token pro Type.

Mit der Konfiguration kann festgelegt werden, welche der ermittelten Daten in den Ergebnis-Vektor aufgenommen werden sollen und wie mit dem Input verfahren werden soll (Ausschluss zu kurzer Wörter, Überführen der Groß- in Kleinbuchstaben).

Graphemizer Component		
Produziert	GraphemeVectorEngine	
Konsumiert	Tokenizer	
Datenbank	Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Min word length	Minimale Länge der zu analysierenden Wörter
	To lowercase	Großbuchstaben werden in Kleinbuchstaben überführt
	Partners to vector	Aufnahme der Minimalpaarpartner in den Vektor
	Contexts to vector	Aufnahme der Minimalpaarkontexte in den Vektor
	Left-right inword positions to vector	Aufnahme der Position des Minimalpaars innerhalb eines Wortes (von links betrachtet) in den Vektor
	Right-left inword positions to vector	Aufnahme der Position des Minimalpaars innerhalb eines Wortes (von rechts betrachtet) in den Vektor
	Count weigth in vector	Gewicht der Häufigkeit eines Graphems im Vektor
Funktion	Detektiert aus dem Input Grapheme durch Minimalpaaranalyse.	

C.2.8 Morphemizer

In den beiden Morphemizern ist die in Kapitel 6.2.1 beschriebene distributionelle Analyse implementiert. Dabei werden aus den Wörtern des Inputs sogenannte Keyword-Trees erzeugt, aus denen sich ermitteln lässt, wie viele verschiedene Vorgänger- und auch Nachfolgerknoten die Zeichenketten des Inputs haben. Die Verteilung der unterschiedlichen Anzahl dieser Knoten kann Rückschlüsse auf Morphemgrenzen geben. Über die Konfiguration wird festgelegt, welche Eigenschaften in dieser Verteilung als Hinweis auf bzw. Evidenz für Morphemgrenzen erwählt werden sollen („scoring“). Ebenfalls konfigurierbar ist, wie viele dieser Scores nötig sind, um eine Morphemgrenze anzunehmen. Mit der Anzahl der Iterationen legt man fest, wie oft der Output der Distributionsanalyse wieder als Input der nächsten Analyse verwendet werden soll, um die Morpheme weiter zerlegen zu können.

Die Morphemizer erzeugen ein Morphemlexikon, in dem auch die Häufigkeiten der ermittelten Morpheme aus dem Input angegeben werden (Rolle Morphemizer). Darüber hinaus bietet er über den SubstitutionTokenAccessAdapter die Möglichkeit, neue, d.h. auch nicht zum ursprünglichen Input gehörende Wörter in Morpheme zu segmentieren (Rolle Tokenizer). Dabei kann gewählt werden, ob man alle Segementierungen oder nur die wahrscheinlichste geliefert haben möchte. Die Wahrscheinlichkeit von Morphemketten ist in diesem Fall eine Funktion über die Häufigkeit der potentiellen Morpheme.

Die SimpleMorphemizer Komponente greift für ihre Analyse ausschließlich auf die distributionell gewonnenen Daten zurück. Die ExtendedMorphemizer Komponente dagegen kann noch mit zusätzlichem Input gefüttert werden, etwa mit Morphemen aus einem manuell erstellten Lexikon oder – wie in der konkreten Implementation vorliegend – durch Cluster über Grapheme, die mit in die Berechnung potentieller Morphemgrenzen einbezogen werden. Der Extended Morphemizer ist ein Beispiel dafür, wie das Mittel der Vererbung auch für Tesla-Komponenten nutzbar ist. Zugleich wird auch der Unterschied zwischen Komponenten (die sich hier durch ihren Input unterscheiden) und Rollen (die über den Output definiert sind und damit für beide Komponenten die gleichen sind) aufgezeigt.

Simple / Extended Morphemizer Component		
Produziert	MorphemeTokenizer, MorphemeLexGenerator	
Konsumiert	Tokenizer, Clusterer (nur ExtendedMorphemizer)	
Datenbank	Tunguska, Hibernate	
Autor	Jürgen Hermes	
Konfiguration	Min word length	Minimale Länge der zu analysierenden Wörter
	Evidence level	Minimale Anzahl der Evidenz-Punkte, die zur Annahme einer Morphemgrenze führen
	Solely maximum splitter	Wort wird in höchstens zwei Morpheme geteilt
	Iterations	Anzahl der Iterationen (bei denen versucht wird, Morpheme des Vorgängerdurchgangs weiter in Morpheme zu zerlegen)
	Include affix position	Unterscheidung von Prä-, In- und Suffixen bei der Morphemanalyse
	Include increasing predecssor counts	Vergabe von Evidenzpunkten für ansteigende Vorgängerwerte
	Include increasing successor counts	Vergabe von Evidenzpunkten für ansteigende Nachfolgerwerte
	Include local max predecessor counts	Vergabe von Evidenzpunkten für lokale Maxima bei Vorgängerwerten
	Include local max successor counts	Vergabe von Evidenzpunkten für lokale Maxima bei Nachfolgerwerten
	Include max predecessor counts	Vergabe von Evidenzpunkten für absolute Maxima bei Vorgängerwerten
	Include max successor counts	Vergabe von Evidenzpunkten für absolute Maxima bei Nachfolgerwerten
	Include local max combined counts	Vergabe von Evidenzpunkten für lokale Maxima kombinierter Vorgänger/Nachfolgerwerte
	Include max combined counts	Vergabe von Evidenzpunkten für absolute Maxi-ma kombinierter Vorgänger/Nachfolgerwerte
	Include change of grapheme-cluster	Vergabe von Evidenzpunkten, wenn Grapheme aus unterschiedlichen Klassen stammen (nur ExtendedMorphemizer).
Funktion	Detektiert Morpheme aus dem gegebenen Input über distributionelle Analyse von Vorgänger/Nachfolgerknoten und – im Falle des Extended Morphemizer – durch Analyse der Clusterzugehörigkeit der Grapheme.	

C.2.9 K-Means++ Clusterer

Diese Komponente ist ein Beispiel für die Nutzung von Fremdbibliotheken in Tesla durch das Wrappen der Funktionalität in einer Tesla-Komponente. Konkret wird hier der `KMeansPlusPlus Clusterer` des `apache.commons.math`-Projektes[7] genutzt. Dieser verteilt die Vektoren der Eingabe auf eine zu spezifizierende Anzahl k Cluster.

K-Means++ Clusterer		
Produziert	WeightedClusterer	
Konsumiert	VectorGenerator	
Datenbank	DB4O	
Autor	Stephan Schwiebert	
Konfiguration	Number of Clusters	Anzahl der Cluster, auf die die Vektoren verteilt werden sollen
	Distance Function	Auswahl des Distanzmaßes
	Max Iterations	Anzahl der maximalen Iterationen des Algorithmus
	Seed	Festlegen der Auswahl der initialen Cluster
Funktion	Clustert die Eingabevektoren in die gewünschte Menge von Clustern mithilfe des kMeans++-Verfahrens.	

C.2.10 NGram Tree Generator

Der `NGramTreeGenerator` erzeugt aus den Sequenzen des Inputs einen generalisierten Suffixbaum, dessen Knoten den Tokens der Eingabe entsprechen. Jeder Knoten enthält darüber hinaus die Information, in welchen Sequenzen er welches Token er repräsentiert. Der Baum kann vorwärts oder rückwärts aufgebaut werden, das Resultat ist dann dementsprechend ein Suffix- oder ein Präfixbaum. Über die Konfiguration kann ferner festgelegt werden, wie tief der Baum maximal wachsen darf, das heißt, bis zu welchem n die n-Gramme in ihm abgebildet werden. Dies ermöglicht es, Speicherüberläufe auch bei der Bearbeitung größerer Korpora zu vermeiden.

7 Siehe `http://commons.apache.org/math`, zuletzt aufgerufen am 11.10.2011

NGram Tree Generator		
Produziert	NGramTreeGenerator	
Konsumiert	AnchoredElements (Sequenzen), AnchoredElements (Tokens)	
Datenbank	Tunguska	
Autor	Stephan Schwiebert	
Konfiguration	Max N-Gram length	Maximale Tiefe des Baums
	Reverse	Erzeugen eines Suffix- (-reverse) oder Präfix-Baum (+reverse)
Funktion	Erzeugt einen generalisierten Suffix- bzw. Präfix-Baum aus den gegebenen Sequenzen. Knoten dieses Baums sind die gegebenen Token-Elemente.	

C.2.11 Pattern Detector

Der `PatternDetector` sucht nach Wiederholungsmustern im übergebenen `NGramTree` und mappt diese auf das Ausgangssignal zurück. Mindestlänge und Mindesthäufigkeit der Muster können über die Konfiguration festgelegt werden. Wird die Mindestlänge auf "1" gesetzt, so liefert die Komponente nur die längsten Muster, die in der spezifizierten Häufigkeit vorliegen, ansonsten werden alle Muster geliefert, die beiden in der Konfiguration spezifizierten Anforderungen genügen.

Pattern Detector		
Produziert	Constituent Tagger	
Konsumiert	AnchoredElements (Sequenzen), AnchoredElements (Tokens), NGramTree	
Datenbank	Tunguska	
Autor	Jürgen Hermes	
Konfiguration	Minimum occurence of pattern	Mindestzahl, in der ein Muster vorkommen muss, um markiert zu werden.
	Minimum length of patterns	Mindestlänge an Einheiten, die ein Muster umspannen muss, um markiert zu werden. Beim Setzen des Wertes auf eine Zahl kleiner zwei werden nur die längsten Muster markiert.
Funktion	Markiert gleiche Abfolgen von Einheiten, die mindestens über spezifizierte Länge und Häufigkeit verfügen.	

Anhang D

Listings

Auf den folgenden Seiten finden sich diejenigen Listings, auf die in Kapitel 3 referenziert wurde und die auf dem begrenzten Platzangebot einer Buchseite darstellbar sind. Alle weiteren Listings, wie die Definition der zentralen Sammlung von Tesla-Rollen (roles.xml) und die XML-Repräsentationen der in dieser Arbeit vorgestellten Experimente finden sich online.[1]

Liste der dort zugänglichen Ressourcen (zu einem Archiv `jh.diss.supp.zip` gepackt):

- Zentrales Verzeichnis der Tesla-Rollen (`roles.xml`), Stand 16.01.2012. Die an den Experimenten dieser Arbeit beteiligten Komponenten wurde in der Version eingefroren, in der sie zum Einsatz kamen (`comps111011.zip`).
- Prozessierte Texte (VIAF, Vergleichstexte, PIII-Chiffren) im Ordner *texts*.
- Experimente zur Ermittlung der statistischen Kennwerte (referenziert in Kapitel 6.1.3) finden sich im Ordner *cipher_ method_ detection*.
- Experimente zur Ermittlung und Analyse von Graphemen und Morphemen (referenziert in Kapitel 6.2.3) finden sich im Ordner *cipher_ key_ detection*.
- Experimente zur Detektion von Mustern (referenziert in Kapitel 6.3.3) finden sich im Ordner *plaintext_ detection*.
- Ein Experiment, welches die Latex-Dateien, die dieser Arbeit zugrundeliegen, einliest und mehrere Filter einsetzt, um Nomen zu extrahieren. Eine solche Abfolge von Komponenten wurde auch verwendet für die Visualisierung der WordCloud[2] auf der Titelseite dieser Arbeit (Ordner *wordcloud_ experiment*).

Experimente und Texte wurden außerdem auf die Social Science Platform *MyExperiment* (siehe Kapitel 7) hochgeladen. Die Experimente wurden dabei in drei sogenannte *Packs* aufgeteilt, eins für die Ermittlung der Chiffriermethode,[3] ein weiteres für die Rekonstruktion der Schlüssel,[4] ein drittes für die Mustersuche im Text.[5]

1 `http://www.phil-fak.uni-koeln.de/fileadmin/spinfo/jhermes/jh.diss.supplements.zip`
2 Das Bild wurde erzeugt von der Software *Cloudio*, die von Stephan Schwiebert entwickelt wurde und die inzwischen auch Teil von Tesla ist (`http://wiki.eclipse.org/Zest/Cloudio`).
3 *Comparative Statistics* Pack, `http://www.myexperiment.org/packs/240.html`
4 *Structural Methods on Cipher Texts* Pack, `http://www.myexperiment.org/packs/242.html`
5 *Pattern Detection with N-Gram Trees* Pack, `http://www.myexperiment.org/packs/243.html`

```
<!--  ******************** -->
<!--  1.1 The "Word" token -->
<!--  ******************** -->
<spre:token name="Word">
   <spre:pattern>
      <spre:startsWith>Letter</spre:startsWith>
      <spre:contains>Alphanumeric</spre:contains>
      <spre:contains>Dot</spre:contains>
      <spre:contains>Hyphen</spre:contains>
      <spre:contains>Slash</spre:contains>
      <spre:contains>Backslash</spre:contains>
      <spre:ambiguity>
         <spre:element>Dot</spre:element>
         <!--  Merge the ambigue element with the previous element's
             belonging WordLevel-element if the condition isAbbreviation is
             met. -->
         <spre:merge type="left">
            <spre:sequence>
               <spre:element>Letter</spre:element>
               <spre:element>Dot</spre:element>
               <spre:element>Divider</spre:element>
            </spre:sequence>
            <!-- The idea is that conditions call methods, defined in
                <spre:condition ...>, to check for special properties   of
                the item in question. The parameters of the respective method
                is String, in this case the Token "Word" that results by
                applying this Merge operation. -->
            <spre:conditions>
               <spre:condition>Abbreviations</spre:condition>
            </spre:conditions>
         </spre:merge>
         <spre:merge type="left">
            <spre:sequence>
               <spre:element>Letter</spre:element>
               <spre:element>Dot</spre:element>
               <spre:element>Divider</spre:element>
               <spre:element>LowerCaseLetter</spre:element>
            </spre:sequence>
         </spre:merge>
         <!--  Merge all three elements to one of the type of the first
             element's WordLevel-element type. -->
         <spre:merge type="leftright">
            <spre:sequence>
               <spre:element>Alphanumeric</spre:element>
               <spre:element>Dot</spre:element>
               <spre:element>Alphanumeric</spre:element>
            </spre:sequence>
         </spre:merge>
      </spre:ambiguity>
   </spre:pattern>
</spre:token>
```

Listing D.1: *Ausschnitt aus der SPre-Konfigurationsdatei für den WordLayer. Zu sehen ist die Definition des Elements "Word".*

```
<set id="3"> <!-- Reihenfolge fuer Darstellung im Tesla-Editor -->
   <name>Ceasar Cipher Generator</name>
   <description>Generates ciphers based on Caesars method</description>
   <template id="cae"/>       <!-- Template fuer Verfahren -->
   <template id="enc"/>       <!-- Anwendungsrichtung -->
   <template id="int_key"/>   <!-- Default-Key -->
   <template id="rep_german"/> <!-- ErsetzungsEH, z.B. Umlaute -->
   <template id="del_ws"/>    <!-- Template loeschen von Leerzeichen -->
</set>

<template id="enc" name="Encipherer" category="Direction">
   <description>Encrypts plain texts to cipher texts</description>
   <value>false</value>
</template>

<template id="int_key" name="Caesar test key" category="Key">
   <description> Value 2 as Key </description>
   <value>2</value>
</template>

<template id="rep_german" name="German replacement characters"
    category="Replacements">
   <description>Characters that should be replaced before
       enciphering</description>
   <reference file="replacements_german.csv"/>
      <!-- Hier Verweis auf eine Datei, in der Replacements abgelegt sind -->
</template>

<template id="del_ws" name="Delete whitespaces" category="Whitespace
    deletion">
   <description>Deletes all whitespace characters from   the substitution
       text</description>
   <value>true</value>
</template>
```

Listing D.2: *Beispiel für ein TemplateSet mit zugehörigen Templates: Auswahl passender Werte für die Caesar-Chiffre*

```
<?xml version="1.0" encoding="UTF-8"?>
<role_system context="generic_roles">

   <role id="de.uni_koeln.spinfo.tesla.rolesystem.presets.roles.Annotator">
      <metadata>
         <name>Annotator</name>
         <description>Base role of every role in Tesla.</description>
      </metadata>
         <produces>
            <analysis>
               <data_object_interface>
                  de.uni_koeln.spinfo.tesla.runtime.persistence.DataObject
               </data_object_interface>
               <input_adapter_interface>
                  [. . .].tesla.annotation.adapter.IAccessAdapter
               </input_adapter_interface>
            </analysis>
         </produces>
   </role>

   <role id="de.uni_koeln.spinfo.tesla.roles.core.AnchoredElementGenerator">
      <metadata>
         <name>Anchored Element Generator</name>
         <description>Generates anchored elements.</description>
      </metadata>
      <produces>
         <analysis>
            <data_object_interface>
               de.uni_koeln.spinfo.tesla.roles.core.data.IAnchoredElement
            </data_object_interface>
            <input_adapter_interface>
               [. . .].tesla.roles.core.access.IAnchoredElementAccessAdapter
            </input_adapter_interface>
            </analysis>
         </produces>
   </role>

   <role id="de.uni_koeln.spinfo.tesla.roles.core.SequenceAnnotator">
      <metadata>
         <name>Sequencer</name>
         <description>Annotates sub sequences of the text with
             labels.</description>
      </metadata>
         <produces>
            <analysis>
               <data_object_interface>
                  de.uni_koeln.spinfo.tesla.roles.core.data.ISubSequence
               </data_object_interface>
               <input_adapter_interface>
                  [. . .].tesla.roles.core.access.ISequenceAccessAdapter
               </input_adapter_interface>
            </analysis>
         </produces>
   </role>
</role_system>
```

Listing D.3: *Die generischen Tesla-Rollen*

Anhang E

Quellen

Die Unterscheidung zwischen Quellen und Sekundärliteratur ist auf dem Gebiet der historischen Kryptologie nicht leicht vorzunehmen, da die Grenze zwischen beiden nicht scharf gezogen werden kann: Viele Schriften setzen sich mit vorangegangenen Chiffren auseinander, führen zugleich aber auch neue Chiffren ein, so z.B. de Vigenère (1586) und Selenus (1624). Aufgrund dieser Unschärfe werden die meisten Schriften, die auch den Quellen hätten zugeordnet werden können, im Literaturverzeichnis aufgeführt. Das betrifft die Universalsprachentwürfe von Dalgano (1661) und Wilkins (1668) genauso wie die Adaption der trithemischen Polygraphia durch de Collange (1561). Als Quellen werden hier nur diejenigen Texte aufgeführt, die innerhalb dieser Arbeit auch im Detail analysiert werden - das Voynich-Manuskript sowie die beiden kryptographischen Werke von Trithemius, dazu Briefe, die sich unmittelbar auf diese Schriften beziehen.

E.1 Voynich-Manuskript

Unbekannter Autor: Manuskript unbekannten Inhalts und unbekannten Ursprungs. Signatur MS 408 an der Yale University, Beinecke Rare Book and Manuskript Library - General Collection of Rare Books and Manuskripts - Medieval and Renaissance Manuskripts. Erstellt mutmaßlich zwischen 1404 und 1622. Faksimiles aller Seiten finden sich unter `http://beinecke.library.yale.edu/dl_crosscollex/getSETS.asp?ITEM=2002046`, zuletzt aufgerufen am 11.10.2011.

Johannes Marcus Marci von Kronland: *Brief an Athanasius Kircher in Rom. Datiert auf den 19. August 1666.* Supplement zum oben genannten MS 408. Faksimile unter `http://www.voynich.nu/extra/img/marci2k.jpg`, Lateinische Transkription und englische Übersetzung unter `http://www.voynich.nu/letters.html#gb39` zuletzt aufgerufen am 11.10.2011.

Georg Baresch: *Brief an Athanasius Kircher in Rom. Datiert auf den 27 April 1639.* Archiv der Pontificia Università Gregoriana in Rom, Signatur APUG 557, fol. 353. Lateinische Transkription und englische Übersetzung unter `http://www.voynich.nu/letters.html#gb39` zuletzt aufgerufen am 11.10.2011.

E.2 Trithemius' kryptographische Schriften

Johannes Trithemius: *Polygraphia libri sex*, Basel: Haselberg 1518 (geschrieben 1506). Faksimiles unter `http://diglib.hab.de/wdb.php?dir=drucke/fb-128` zuletzt aufgerufen am 11.10.2011.

Johannes Trithemius: *Steganographia [...]. Preafixa est hvic operi sva clavis*, Frankfurt: Berner 1606 (geschrieben 1499). Faksimiles unter `http://diglib.hab.de/wdb.php?dir=drucke/fb-243` zuletzt aufgerufen am 11.10.2011.

Johannes Trithemius: *"Apologetische Vorrede" zur "Steganographia"*, in: Heinrich Cornelius Agrippa's von Nettesheim Magische Werke sammt den geheimnisvollen Schriften des Petrus von Abano, [... und] Abt Tritheim von Spanheim [... I. Zum ersten Male vollstandig in's Deutsche iibersetzt. Bd 1-5. 4. Aufl. (Berlin 1924); Nachdruck der 5 Bücher in 2 Banden Schwarzenburg: Ansata-Verlag 1979; hier Bd 2, 5. Buch, S. 304. - Lateinischer Wortlaut der "Praefatio" in: Steganographia. Hoc est: Ars per occvltam scriptorvm animi svi volvntatem absentibvs aperiendi certa [...]. Praefixa est hvic operi sva clavis [...]. Frankfurt: Berner 1606.

Johannes Trithemius: *Brief an Arnold Bostius in Gent. Datiert auf den 25. März 1499.* Paris, Bibliothek Mazarine, Signatur 1565[1308], zuletzt gedruckt bei Paulus Volk: Abt Johannes Trithemius, in: Rheinische Vierteljahresblätter 27 (1962), S. 42ff.

Literaturverzeichnis

Adkins, L. & R. Adkins: 2000, *The Keys of Egypt: The Obsession to Decipher Egyptian Hieroglyphs*, HarperCollins, London.

Albrecht, J.: 2002, 'Der Strukturalismus in der Sprachwissenschaft: Erbe und Auftrag', in E. Kennosuke (ed.), *Linguistik jenseits des Strukturalismus*, Narr, Tübingen, pp. 145–154.

Arnold, K.: 1971, *Johannes Trithemius (1462-1516)*, Schöningh, Würzburg.

Barga, R. & D. Gannon: 2006, 'Scientific versus Business Workflows', in I. Taylor, E. Deelman & D. Gannon (eds.), *Workflows for e-Science: Scientific Workflows for Grids*, chap. 2, Springer, New York, pp. 9–16.

Bauer, F. L.: 2000, *Entzifferte Geheimnisse: Methoden und Maximen der Kryptologie, 3. Auflage*, Springer, Berlin, Heidelberg.

de Beaugrande, R.-A. & W. U. Dressler: 1981, *Einführung in die Textlinguistik*, Niemeyer, Tübingen.

Benden, C.: 2005, 'Automated Detection of Morphemes Using Distributional Measurements', in C. Weihs & W. Gaul (eds.), *Classification - The Ubiquitous Callenge*, Springer, Berlin, pp. 490–497.

Benden, C.: 2006, 'Bootstrapping an Unsupervised Morphemic Analysis', in M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nürnberger & W. Gaul (eds.), *From Data and Information Analysis to Knowledge Engineering*, Springer, Berlin, pp. 318–325.

Benden, C. & J. Hermes: 2004, 'Präprozessierung mit Nebenwirkungen: Dynamische Annotation', in E. Buchberger (ed.), *Beiträge zur 7. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS)*, Riegelnik, Wien, pp. 25–28.

Bennett, W. R.: 1976, *Scientific and Engineering Problem-Solving with the Computer*, Prentice Hall PTR, Upper Saddle River, NJ.

Bergman, M.: 2001, 'White Paper: The Deep Web. Surfacing Hidden Value', *The Journal of Electronic Publishing* **7**(1).

Bird, S. & M. Liberman: 1999, 'A Formal Framework for Linguistic Annotation', *CoRR* **cs.CL/9903003**.

Bloomfield, L.: 1926, 'A Set of Postulates for the Science of Language', *Language* **2**, 153–164.

Bloomfield, L.: 1935, *Language*, Allen & Unwin, London.

Burckhardt, J. & W. W. Goetz: 1922, *Die Kultur der Renaissance in Italien, ein Versuch von Jacob Burckhardt*, A. Kroner, Stuttgart, 3. Aufl. Neudruck der Urausgabe, durchgesehen von Walter Goetz.

Cavalli-Sforza, L.: 1997, 'Genes, Peoples, and Languages', *Proceedings of The National Academy of Sciences* **94**, 7719–7724.

Cherry, C.: 1963, *Kommunikationsforschung - eine neue Wissenschaft*, Fischer, Frankfurt a. M.

Chomsky, N.: 1957, *Syntactic Structures*, Mouton and Co, The Hague.

Chomsky, N.: 1965, *Aspects of the Theory of Syntax*, M.I.T. Press, Cambridge, Massachusetts.

Chomsky, N.: 1981, *Lectures on Government and Binding. The Pisa Lectures*, Foris Publications, Dordrecht.

Chomsky, N.: 1995, *The Minimalist Program*, MIT Press, Cambridge, Massachusetts.

Christmann, U.: 2000, 'Aspekte der Textverarbeitungsforschung', in K. Brinker, G. Anthos, W. Heinemann & S. Sager (eds.), *Text- und Gesprächslinguistik Bd. I*, Mouton de Gruyter, Berlin, New York, pp. 113–122.

Chung, M.-H. *et al.*: 2006, 'Investigation Committee Report', Tech. rep., Seoul National University.

de Collange, G.: 1561, *Polygraphie, et Vniverselle esciture Cabalistique de M.I. Trithe-me Abbé*, Kerner, Paris.

Couturat, L. & L. Leau: 1903, *Histoire de la langue universelle*, Hachette, Paris.

Cunningham, H. & K. Bontcheva: 2006, 'Computational Language Systems, Architectures', in K. Brown, A. H. Anderson, L. Bauer, M. Berns, G. Hirst & J. Miller (eds.), *The Encyclopedia of Language and Linguistics*, second edn., Elsevier, München.

Currier, P.: 1976, 'Papers on the Voynich Manuskript', `http://www.voynich.nu/extra/curr_main.html`, zuletzt aufgerufen: 11.10.2011.

Dalgano, G.: 1661, *Ars Signorum*, Eigendruck, London, Nachdruck Menston: The Scolar Press 1968.

Davidsson, C.: 1959, 'Johannes Tritemius' Polygraphia als tschechisches Lehrbuch. Cod. Slav. der Universitätsbibliothek in Uppsala', *Scando-Slavica* **5**, 148–164.

De Roure, D., C. Goble, S. Aleksejevs, S. Bechhofer, J. Bhagat, D. Cruickshank, P. Fisher, N. Kollara, D. Michaelides, P. Missier, D. Newman, M. Ramsden, M. Roos, K. Wolstencroft, E. Zaluska & J. Zhao: 2010, 'The Evolution of myExperiment', in *Sixth IEEE e–Science conference (e-Science 2010)*, Brisbane, Australia, in Press.

De Roure, D., C. Goble & R. Stevens: 2009, 'The design and realisation of the Virtual Research Environment for social sharing of workflows', *Future Generation Computer Systems* **25**(5), 561–567.

Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas & R. A. Harshman: 1990, 'Indexing by Latent Semantic Analysis', *Journal of the American Society of Information Science* **41**(6), 391–407.

D'Imperio, M.: 1978a, 'An Application of Cluster Analysis and Multidimensional Scaling to the Question of "Hands" and "Languages" in the Voynich Manuscript', *National Security Agency Technical Journal* **23**(3), 59–75.

D'Imperio, M.: 1978b, *The Voynich Manuscript–An Elegant Enigma*, Aegean Park Press, Langley.

Dougherty, R.: 1994, *Natural Language Computing an English Generative Grammar in PROLOG*, Erlbaum, Hillsdale, N.J.

Dubin, D.: 2004, 'The most influential paper Gerard Salton never wrote', *Library Trends* **52**(4), 748–764.

Dunning, T.: 1994, 'Statistical Identification of Language', Tech. rep., Computing Research Lab, New Mexico State University.

Eis, G.: 1962, *Mittelalterliche Fachliteratur*, Metzler, Stuttgart.

Embach, M.: 2003, *Die Schriften Hildegards von Bingen*, Akademie Verlag, Berlin.

Ernst, T.: 1996, 'Schwarzweiße Magie. Der Schlüssel zum dritten Buch der Steganographia des Trithemius', *Daphnis* (25), 1–205.

Ernst, T.: 2001, 'Anatomie einer Fälschung: Johannis Thrithemij [...] Steganographiae Lib 3. cum Clave, tàm generalj, quàm specialj [...] M.D.XXI', *Daphnis* (20), 513–595.

Feely, J.: 1943, *Roger Bacon's Cipher: The Right Key Found,*, Eigenverlag, Rochester, NY.

Feldman, R. & J. Sanger: 2006, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*, Cambridge University Press, Cambridge, MA.

Fillmore, C.: 1992, ''Corpus-Linguistics' vs. 'Computer-Aided Armchair Linguistics'', in J. Svartvik (ed.), *Directions in corpus linguistics : proceedings of Nobel Symposium 82, Stockholm, 4 - 8 August 1991*, Mouton de Gruyter, Berlin [u.a.], pp. 35–60.

Foster, I.: 2002, 'What is the Grid? A Three Point Checklist', *GRIDtoday* **1**.

Friedman, W.: 1922, 'The Index of Coincidence and its Applications in Cryptology', Tech. Rep. 22, Riverbank Laboratories, Department of Ciphers, Geneva, Illinois.

Friedman, W. & E. Friedman: 1959, 'Acrostics, Anagrams, and Chaucer', *Philological Quaterly* **38**.

Fuchs, R., C. Meinert & J. Schrempf: 2001, *Pergament :Geschichte - Material - Konservierung - Restaurierung*, Siegl, München.

Geison, G.: 1995, *The Private Science of Louis Pasteur.*, Princeton University Press, Princeton, NJ.

Ghemawat, S., H. Gobioff & S. Leung: 2003, 'The Google File System', in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, ACM, New York, NY, pp. 29–43.

Götz, T. & O. Suhre: 2004, 'Design and Implementation of the UIMA Common Analysis System', *IBM Syst. J.* **43**(3), 476–489.

Grafton, A.: 2002, *Leon Battista Alberti*, Berlin Verlag, Berlin.

Grzybek, P.: 2006, 'History and Methodology of Word Length Studies. The State of the Art', in P. Grzybek (ed.), *Contributions to the Science of Text and Language. Word Length Studies and Related Issues*, Springer, Dordrecht, NL, pp. 15–90.

van Gurp, J. & J. Bosch: 2002, 'Role-Based Component Engineering', in M. L. I. Crnkovic (ed.), *Building Reliable Component-based Systems*, Artech House.

Gusfield, D.: 1997, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, New York, NY.

Gutknecht, J.: 1985, 'Concepts of the Text Editor Lara', *Commun. ACM* **28**(9), 942–960.

Guy, J.: 1996, 'The Frogguy Transliteration System', `http://www.voynich.net/reeds/tutorial.html`, zuletzt aufgerufen: 11.10.2011.

Guy, J. B. M.: 1991a, 'Statistical Properties of two Folios of the Voynich manuscript', *Cryptologia* **15**(3), 207–218.

Guy, J. B. M.: 1991b, 'Vowel Identification: an old (but good) Algorithm', *Cryptologia* **15**(3), 258–262.

Günther, H.: 1988, *Schriftliche Sprache. Strukturen geschriebener Wörter und ihre Verarbeitung beim Lesen*, Niemeyer, Tübingen.

Hagenbruch, A.: 2010, 'Flache Satzverarbeitung', in K. Carstensen, C. Ebert, S. Jekat, C. Ebert, H. Langer & R. Klabunde (eds.), *Computerlinguistik und Sprachtechnologie. Eine Einführung - 3. Auflage*, Elsevier, München, pp. 264–279.

Hamlet, D., D. Mason & D. Woit: 2001, 'Theory of Software Reliability Based on Components', in *Proceedings ICSE '01*, IEEE Computer Society, pp. 361–370.

Harris, Z.: 1951, *Methods in Structural Linguistics*, University of Chicago Press, Chicago.

Harris, Z.: 1952, 'Discourse Analysis', *Language* **28**, 1–30.

Harris, Z.: 1954, 'Distributional Structure', *Word* **10**, 146–162.

Heinen, N.: 2011, 'Falsches Spiel', *Technology Review* **2011**(01), 52–57.

Hermes, J. & C. Benden: 2005, 'Fusion von Annotation und Präprozessierung als Vorschlag zur Behandlung des Rohtextproblems', in B. Fisseni, H.-C. Schmitz, B. Schröder & P. Wagner (eds.), *Sprachtechnologie, mobile Kommunikation und linguistische Ressourcen. Beiträge zur GLDV-Tagung 2005 in Bonn* (*Sprache, Sprechen und Computer* 8), Lang, Frankfurt am Main, pp. 78–90.

Hermes, J. & S. Schwiebert: 2010, 'Classification of Text Processing Components: The Tesla Role System', in A. Fink, B. Lausen, W. Seidel & A. Ultsch (eds.), *Advances in Data Analysis, Data Handling and Business Intelligence*, Studies in Classification, Data Analysis, and Knowledge Organization, Springer, Berlin, Heidelberg, pp. 285–294.

Heyer, G., U. Quasthoff & T. Wittig: 2006, *Text Mining: Wissensrohstoff Text*, W3L GmbH, Witten Herdecke.

Hochmuth, M., A. Lüdeling & U. Leser: 2008, ' Simulating and Reconstructing Language Change ', Tech. Rep. 220, Humboldt-Universität zu Berlin, Institut für Informatik.

Hull, D., K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li & T. Oinn: 2006, 'Taverna: a Tool for Building and Running Workflows of Services', *Nucleic acids research* **34**(Web Server issue), 729–732.

Hurych, J.: 2007, 'The New Signature of Horčický and the Comparison of them All', `http://hurontaria.baf.cz/CVM/b12.htm`, zuletzt aufgerufen: 11.10.2011.

Hurych, J.: 2008a, 'How Many Hands Wrote the VM', `http://hurontaria.baf.cz/CVM/a17.htm`, zuletzt aufgerufen: 11.10.2011.

Hurych, J.: 2008b, 'The VM Manuscript Letter Frequency', `http://hurontaria.baf.cz/CVM/a26.htm`, zuletzt aufgerufen: 11.10.2011.

Hutchins, J.: 1986, *Machine Translation: Past, Present, Future*, Ellis Horwood Series in Computers and their Applications, Ellis Horwood, Chichester.

Hřebíček, L. & G. Altmann: 1993, 'Prospect of Text Linguistics', in L. Hřebíček & G. Altmann (eds.), *Quantitative Text Analysis*, Wissenschaftlicher Verlag, Trier, pp. 1–28.

Jojic, V., D. Heckerman, C. Kadie, C. Meek, C. Moore, M. John & S. Mallal: 2005, 'HLA-driven Optimization of an HIV Vaccine Immunogen', `http://www.retroconference.org/2005/cd/Abstracts/25449.htm`, zuletzt aufgerufen: 11.10.2011.

Jurafsky, D. & J. H. Martin: 2008, *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*, 2 edn., Prentice Hall, Upper Saddle River, NJ.

Kahn, D.: 1996, *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*, Scribner, New York.

Kasiski, F. W.: 1668, *Geheimschriften und die Dechiffrirkunst*, Mittler und Sohn, Berlin.

Kay, L.: 2000, *Who wrote the Book of Life*, Stanford University Press, Stanford, CA.

Kay, M.: 2004, 'Substring Alignment Using Suffix Trees', *Computational Linguistics and Intelligent Text Processing* , 275–282.

Kennedy, G. & R. Churchill: 2004, *The Voynich manuscript : the Unsolved Riddle of an Extraordinary Book which has Defied Interpretation of Centuries*, Orion, London, deutsche Übersetzung: 'Der Voynich-Code: Das Buch, das niemand lesen kann', Rogner & Bernhard, Berlin (2005).

Kersten, M.: 2007, *Focusing Knowledge Work with Task Context*, Ph.D. thesis, University of British Columbia, Vancouver, Canada.

Kircher, A.: 1663, *Polygraphia nova et universalis*, Varesius, Rom.

Klavans, J. L. & P. Resnik (eds.): 1996, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, MIT Press, Cambridge, MA.

Kniele, R.: 1889, *Das erste Jahrzehnt der Weltsprache Volapük*, A. Schoy, Ueberlingen.

Knuth, D.: 1984, 'Literate programming', *The Computer Journal* **22**(2), 97–111.

Koch, W.: 1966, 'Einige Probleme der Textanalyse', *Lingua* **16**, 383–398.

Kokol, P., V. Podgorelec, M. Zorman, T. Kokol & T. Njivar: 1999, 'Computer and Natural Language Texts - A Comparison Based on Long-Range Correlations', *JASIS* **50**(14), 1295–1303.

Kouklakis, M., K. Markopoulos, R. Matthew & D. Hunston: 2007, 'Corpus Manager: A Tool for Multilingual Corpus Analysis', in *Proceedings from Corpus Linguistics Conference*.

Kraus, H. P.: 1978, *A Rare Book Saga : the Autobiography of H. P. Kraus*, Putnam, New York.

Kucharczik, K.: 2009, 'Strukturalismus', in J. Schneider (ed.), *De Gruyter Lexikon: Methodengeschichte der Germanistik*, de Gruyter, Berlin, New York, pp. 679–700.

Kuhn, T. S.: 1970, *The Structure of Scientific Revolutions*, University of Chicago Press, Chicago.

Kuper, M.: 1998, *Johannes Trithemius — der schwarze Abt*, Zerling, Berlin.

Lalande, J.-Y.: 1997, *Verbstellung im Deutschen und Französischen*, no. 365 in Linguistische Arbeiten, Niemeyer, Tübingen.

Laver, M. & K. Benoit: 2003, 'Extracting Policy Positions from Political Texts Using Words as Data', *American Political Science Review* **97**(2), 311.

Leech, G.: 1993, 'Corpus Annotation Schemes', *Lit Linguist Computing* **8**(4), 275–281.

Lehmann, P.: 1961, *Merkwürdigkeiten des Abtes Johannes Trithemius*, Verlag der Bayerischen Akademie der Wissenschaften, München.

Li, W.: 1992, 'Random Texts Exhibit Zipf's-Law-Like Word Frequency Distribution', *IEEE Transactions on Information Theory* , 1842–1845.

Lobin, H.: 2001, *Informationsmodellierung in SGML und XML*, Springer, Berlin, Heidelberg.

Lobin, H. & L. Lemnitzer: 2003, 'Text(e) technologisch', in H. Lobin & L. Lemnitzer (eds.), *Texttechnologie. Perspektiven und Anwendungen.*, Stauffenburg, Tübingen, pp. 1–9.

Lund, K. & C. Burgess: 1996, 'Producing High-Dimensional Semantic Spaces from Lexical Co-Occurrence', *Behavior Research Methods Instruments and Computers* **28**(2), 203–208.

Manly, J.: 1931, 'Roger Bacon and the Voynich Manuscript.', *Speculum* (6), 345–391.

Manning, C., P. Raghavan & H. Schütze: 2008, *Introduction to Information Retrieval*, Cambridge University Press.

Manning, C. D. & H. Schütze: 1999, *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA.

McCarthy, P.: 2005, 'An Assessment of the Range and Usefulness of Lexical Diversity Measures and the Potential of the Measure of Textual, Lexical Diversity (MTLD)', Dissertation, University of Memphis.

McCracken, G.: 1948, 'Athanasius Kircher's Universal Polygraphy', *Isis* **39**(4), 215–228.

McEnery, T.: 2003, 'Corpus Linguistics', in R. Mitkov (ed.), *The Oxford Handbook of Computational Linguistics*, Oxford Handbooks in Linguistics, Oxford University Press, pp. 448–463.

McEnery, T. & A. Wilson: 1996, *Corpus Linguistics*, Edinburgh University Press, Edinburgh.

Mikheev, A.: 2003, 'Text Segmentation', in R. Mitkov (ed.), *Oxford Handbook of Computational Linguistics*, University Press, Oxford, pp. 201–219.

Morawietz, F. & U. Mönnich: 2004, 'Formale Grundlagen', in H. Lobin & L. Lemnitzer (eds.), *Texttechnologie - Perspektiven und Anwendungen*, Stauffenburg, Tübingen, pp. 109–142.

Agrippa von Nettesheim, H.: 1533, *De occulta philosophia*, Soter, Köln, Faksimiles des ältesten Koelner Druckes, hrsg. und erl. von Karl Anton Nowotny, Graz: Akad. Druck- u. Verl.-Anst. 1967.

von Neumann, J.: 1945, 'First Draft of a Report on the EDVAC', Tech. rep., University of Pennsylvania.

Newbold, W. & R. Kent: 1928, *The cipher of Roger Bacon. By William Romaine Newbold, edited with foreword and notes by Roland Grubb Kent*, University of Pennsylvania Press, Philadelphia.

Nguyen, T.: 2008, 'Freedom to Research: Keeping Scientific Data Open, Accessible, and Interoperable', Tech. rep., Science Commons, `http://sciencecommons.org/wp-content/uploads/freedom-to-research.pdf`, zuletzt aufgerufen: 11.10.2011.

Nirenberg, M. & H. Matthaei: 1961, 'The Dependence Of Cell-Free Protein Synthesis In E. coli Upon Naturally Occurring Or Synthetic Polyribonucleotides', in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 47(10), pp. 1588 – 1602.

Oyama, S.: 1985, *The ontogeny of information : developmental systems and evolution*, Cambridge University Press, Cambridge, MA.

Pearson, H.: 2006, 'Genetic information: Codes and enigmas', *Nature* , 259–261.

Pedersen, T.: 2008, 'Empiricism Is Not a Matter of Faith', *Computational Linguistics* **34**(3), 465–470.

Peng, C.-K., S. Buldyrev, A. Goldberger, S. Havlin, F. Sciotino, M. Simons & H. Stanley: 1992, 'Long-Range Correlations in Nucleotid Sequences', *Nature* **356**, 168–170.

Pierce, J. & J. C. et al.: 1966, 'Language and Machines — Computers in Translation and Linguistics. ALPAC report', Tech. rep., National Academy of Sciences, National Research Council, Washington, DC.

Pommerening, K.: 2008, 'Kryptoanalyse der Enigma I nach Rejewski', `http://www.staff.uni-mainz.de/pommeren/Kryptologie/Klassisch/4a_ZylRot/AnalyseEnigma2.html`, zuletzt aufgerufen: 11.10.2011.

Popper, K.: 1935, *Logik der Forschung*, Julius Springer Verlag, Wien.

Raible, W.: 2001, 'Linguistics and Genetics: Systematic Parallels', in M. Haspelmath, E. König, W. Oesterreicher & W. Raible (eds.), *Language Typology and Language Universals - Sprachtypologie und sprachliche Universalien - La Typologie des langues et les universaux linguistiques. An International Handbook - Ein internationales Handbuch - Manuel international* (*Handbücher zur Sprach- und Kommunikationswissenschaft* 20), de Gruyter, Berlin, New York, pp. 103–23.

Raymond, E.: 2001, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, Inc., Sebastopol, CA.

Reeds, J.: 1998, 'Solved: The Ciphers in Book III of Trithemius's Steganographia', Draft, angenommen von der Cryptologia, später überarbeitet. `http://www.dtc.umn.edu/~reedsj/trit.pdf` zuletzt abgerufen am 11.10.2011.

Reeds, J.: 1999, 'Breakthrough in Renaissance Cryptography. A Book Review', *Cryptologia* **23**(1), 59–62.

Rees, J.: 2010, 'Recommendations for independent scholarly publication of data sets', Tech. rep., Creative Commons, San Francisco, CA.

Rivest, R.: 1992, 'The MD5 Message-Digest Algorithm', RFC 1321 (Informational), `http://www.ietf.org/rfc/rfc1321.txt`, zuletzt aufgerufen am 11.10.2011.

Rolshoven, J.: 1987, 'LPS. Eine linguistische Programmiersprache', in U. Klenk, P. Scherber & M. Thaller (eds.), *Computerlinguistik und philologische Datenverarbeitung*, Olms, Hildesheim, pp. 115–129.

Rolshoven, J.: 1991, 'Lexikalisches Wissen in der maschinellen Übersetzung', in P. Blumenthal, G. Rovere & C. Schwarze (eds.), *Lexikalische Analyse romanischer Sprachen*, no. 353 in Linguistische Arbeiten, Niemeyer, Tübingen, pp. 85–100.

Rugg, G.: 2004, 'An Elegant Hoax? A Possible Solution to the Voynich Manuscript', *Cryptologia* **28**(1), 31–46.

Santifaller, L.: 1932, 'Über südliches und nördliches Pergament', *Der Schlern* (13), 458–463.

Sassoon, G. T.: 1992, 'The application of Sukhotin's algorithm to certain non-English languages', *Cryptologia* **16**(2), 165–173.

de Saussure, F.: 1916, *Cours de linguistique générale*, Payot, Paris.

Schenkel, A., J. Zhang & Y.-I. Zhang: 1992, 'Long Range Correlations in Human Writings', *Fractals* **1**(1), 47–57.

Schinner, A.: 2007, 'The Voynich Manuscript: Evidence of the Hoax Hypothesis', *Cryptologia* **31**(2), 95–107.

Schmeh, K.: 2010, 'Neue Datierung des Voynich-Manuskripts sorgt für Aufsehen', *Telepolis* `http://www.heise.de/tp/r4/artikel/31/31971/1.html`, zuletzt aufgerufen: 11.10.2011.

Schmidt, J.: , 'Hash mich, die zweite. Konsequenzen der erfolgreichen Angriffe auf MD5', `http://heise.de/-270106`, zuletzt aufgerufen: 11.10.2011.

Schmitz, C. & B. Zucker: 2003, *Wissensmanagement*, Metropolitan, Regensburg, Berlin.

Schmitz, U.: 2000, 'Statistische Methoden der Textlinguistik', in K. Brinker, G. Antos, W. Heinemann & S. Sager (eds.), *Text- und Gesprächslinguistik / Linguistics of Text and Conversation* (*Handbücher zur Sprach- und Kommunikationswissenschaft* 16), Walter de Gruyter, Berlin & New York, pp. 196–199.

Schwerdtfeger, E.: 2006, 'Die seltsamen Seiten', `http://voynich.tamagothi.de/2006/05/07/die-seltsamen-seiten/`, zuletzt aufgerufen: 11.10.2011.

Schwiebert, S.: 2011, 'Ein Software Framework für linguistische Anwendungen', Dissertation (MS), Universität zu Köln, Institut für Linguistik - Sprachliche Informationsverarbeitung.

Scott, D.: 2008, 'Voynich Manuscript Pictures', `http://www.apprendre-en-ligne.net/crypto/bibliotheque/Voynich/vms/vmspictures.html`, zuletzt aufgerufen: 11.10.2011.

Selenus, G.: 1624, *Cryptomenytices et Cryptographiæ libri IX*, Gebrüder Stern, Lüneburg, (= Herzog August d. J. von Braunschweig-Lüneburg).

Shannon, C. & W. Weaver: 1949, *The Mathematical Theory of Communication*, University of Illinois Press, Chicago.

Shumaker, W.: 1982, *Renaissance curiosa: John Dee's conversations with angels, Girolamo Cardano's horoscope of Christ, Johannes Trithemius and cryptography, George Dalgarno's Universal language*, Center for Medieval and Early Renaissance Studies, Binghamton, N.Y.

Sieburg, H.: 1991, 'Physiological Studies in Silico', in L. Nadel & D. Stein (eds.), *1990 Lectures in Complex Systems*, vol. 12, Addison-Wesley, pp. 321—-342.

Sinclair, J.: 1992, 'The Automatic Analysis of Corpora', in J. Svartvik (ed.), *Directions in Corpus Linguistics*, Mouton de Gruyter, Berlin, New York.

Singh, S.: 1999, *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography*, Doubleday, New York, NY.

Sperberg-McQueen, C. & L. Burnard: 2002, 'The Text Encoding Initiative Guidelines (P4)', Tech. rep., Text Encoding Initiative, Oxford, Providence, Charlottesville, Bergen.

Stallings, D.: 1998, 'Understanding the Second-Order Entropies of Voynich Text', `http://ixoloxi.com/voynich/mbpaper.htm`, zuletzt aufgerufen: 11.10.2011.

von Stieler, K.: 1673, *Teutsche Sekretariat-Kunst*, Hofmann, Nürnberg.

Stolfi, J.: 1997a, 'A Prefix-Midfix-Suffix Decomposition of Voynichese Words', `http://www.ic.unicamp.br/~stolfi/voynich/97-11-12-pms/`, zuletzt aufgerufen: 11.10.2011.

Stolfi, J.: 1997b, 'Generalized Chinese Theory', `http://www.ic.unicamp.br/~stolfi/voynich/97-11-23-tonal/`, zuletzt aufgerufen: 11.10.2011.

Stolfi, J.: 1998, 'Reeds/Landini's Interlinear File in EVA, Version 1.6e6', `http://www.ic.unicamp.br/~stolfi/voynich/98-12-28-interln16e6/`, zuletzt aufgerufen: 11.10.2011.

Stolfi, J.: 1999, 'OKOKOKO: The Fine Structure of Voynichese Words', `http://www.dcc.unicamp.br/~stolfi/voynich/Notes/017/Note-017.html`, zuletzt aufgerufen: 11.10.2011.

Stolfi, J.: 2000, 'On the VMS Word Length Distribution', `http://www.ic.unicamp.br/~stolfi/voynich/00-12-21-word-length-distr/`, zuletzt aufgerufen: 11.10.2011.

Stolfi, J.: 2002, 'Chinese Theory Redux: Comparing the VMS and East Asian word length distributions', http://www.ic.unicamp.br/~stolfi/voynich/02-01-18-chinese-redux/, zuletzt aufgerufen: 11.10.2011.

Storrer, A.: 2004, 'Text und Hypertext', in H. Lobin & L. Lemnitzer (eds.), *Texttechnologie - Perspektiven und Anwendungen.*, Stauffenburg, Tübingen, pp. 13–50.

Strasser, G.: 1988a, 'Herzog Augusts Handbuch der Kryptographie: Apologie des Trithemius und wissenschaftliches Sammelwerk', in P. Raabe (ed.), *Wolfenbütteler Beiträge: Aus den Schätzen der Herzog August Bibliothek*, vol. 8, Klostermann, Frankfurt/Main, pp. 99–120.

Strasser, G.: 1988b, *Lingua Universalis: Kryptologie und Theorie der Universalsprachen im 16. und 17. Jahrhundert*, no. 38 in Wolfenbütteler Forschungen, Harrassowitz, Wiesbaden.

Strong, L.: 1945, 'Anthony Askham, the Author of the Voynich Manuscript', *Science* (101), 608–609.

Szyperski, C.: 1998, *Component Software*, Addison-Wesley.

Taylor, I., E. Deelman & D. Gannon: 2006, *Workflows for e-Science: Scientific Workflows for Grids*, Springer.

Tillmann, H.: 1997, 'Eight Main Differences between Collections of Written and Spoken Language Data', Tech. rep., FIPKM no 35, Institut für Phonetik und Sprachliche Kommunikation, Universität München.

Tiltman, J.: 1951, 'Transcription of a few Pages of Voynich', http://www.voynich.net/reeds/docs/tiltman.txt, zuletzt aufgerufen: 11.10.2011.

Tiltman, J.: 1967, 'The Voynich Manuscript, The Most Mysterious Manuscript in the World', Tech. rep., NSA Technical Journal 12.

Titus, S. L., J. A. Wells & L. J. Rhoades: 2008, 'Repairing Research Integrity', *Nature* **453**(7198), 980–982.

Vater, H.: 2001, *Einführung in die Textlinguistik. Struktur und Verstehen von Texten*, UTB, Stuttgart.

DE VIGENÈRE, B.: 1586, *Traité des chiffres, ov secretes mainieres d'escrire*, l'Anglier, Paris.

WATSON, J.: 1980, *The Double Helix*, Norton, New York.

WATSON, J. & F. CRICK: 1953, 'Molecular Structure for Desoxyribose Nucleic Acid', *Nature* **171**, 737–738.

WATZLAWICK, P., J. H. BEAVIN & D. D. JACKSON: 2007, *Menschliche Kommunikation. Formen Störungen Paradoxien*, Huber, Bern.

WEAVER, W.: 1949, 'Translation', in N. Locke & A. Booth (eds.), *Machine translation of languages: fourteen essays*, Technology Press of the MIT, New York, pp. 15–23.

WIENER, N.: 1948, *Cybernetics or Control and Communication in the Animal and the Machine*, John Wiley & Sons Inc., New York.

WILKINS, J.: 1668, *An Essay Towards a Real Character*, Gellibrand, London, Nachdruck Menston: The Scolar Press 1968.

WILLIAMS, R. L.: 1999, 'A Note on the Voynich Manuscript', *Cryptologia* **23**(4), 305–309.

WIMMER, G.: 2005, 'The Type-Token Relation', in R. Köhler, G. Altmann & R. Piotrowski (eds.), *Quantitative Linguistik / Quantitative Linguistics* (*Handbücher zur Sprach- und Kommunikationswissenschaft* 27), de Gruyter, pp. 361–368.

VAN ZAANEN, M.: 2000, 'ABL: Alignment-Based Learning', in *In COLING 18*, pp. 961–967.

ZAMENHOF, L.: 1887, *Internationale Sprache. Vorrede und vollständiges Lehrbuch.*, Gebethner & Wolff, Warschau, veröffentlicht unter dem Pseudonym "Doktoro Esperanto".

ZANDBERGEN, R.: 2010, 'Origin of the Manuscript', `http://voynich.nu/origin.html`, zuletzt aufgerufen: 11.10.2011.

ZANDBERGEN, R.: 2011a, 'Analysis of the Text', `http://voynich.nu/analysis.html`, zuletzt aufgerufen: 11.10.2011.

ZANDBERGEN, R.: 2011b, 'Analysis of the Writing (Script)', `http://www.voynich.nu/writing.html`, zuletzt aufgerufen: 11.10.2011.

Zandbergen, R.: 2011c, 'From Digraph Entropy to Word Entropy in the Voynich MS', http://www.voynich.nu/extra/wordent.html, zuletzt aufgerufen: 11.10.2011.

Zandbergen, R.: 2011d, 'History of the Manuscript', http://voynich.nu/history.html, zuletzt aufgerufen: 11.10.2011.

Zandbergen, R.: 2011e, 'Past Research and Proposed Solutions', http://voynich.nu/solvers.html, zuletzt aufgerufen: 11.10.2011.

Zandbergen, R. & G. Landini: 2000, 'EVA Alphabet', http://www.voynich.nu/extra/eva.html, zuletzt aufgerufen: 11.10.2011.

Zipf, G.: 1935, *The Psycho-Biology of Language. An Introduction to Dynamic Philology*, M.I.T. Press, Cambridge, Mass.

Zipf, G. K.: 1949, *Human Behavior and the Principle of Least Effort. An Introduction to Human Ecology*, Addison-Wesley, Cambridge, Mass.