rtfdtohtml

Generierung von Web-Seiten aus Rich Text

Diplomarbeit

von
Elmar Ludwig
Universität Osnabrück

1999

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Überblick

In den letzten Jahren hat die rasant steigende Verbreitung des *World Wide Web* dem früher ausschließlich für wissenschaftliche Zwecke genutzten Internet eine unerwartete Popularität verschafft. Dieser Durchbruch wurde erst möglich durch die Entwicklung einer standardisierten und vielseitigen Dokument-Beschreibungssprache, der *Hypertext Markup Language*, kurz HTML, die erstmals den Aufbau ansprechender Oberflächen ermöglichte. Eine genaue Definition der Sprache befindet sich in [HTML 97].

Trotz der inzwischen immer zahlreicher gewordenen HTML-Editoren gibt es leider immer noch keine wirklich zufriedenstellende Lösung für das Erstellen von eigenen Dokumenten in HTML. Programme ohne aufwendige grafische Aufbereitung des Dokuments bieten keine wirkliche Hilfe bei der Erstellung, und Programme mit einer aufwendigen Benutzerschnittstelle sind häufig kompliziert zu bedienen oder schränken den Autor in seinen Möglichkeiten zu sehr ein. Traditionell wurde (und wird auch heute noch) ein großer Teil dieser Dokumente ohne weitere Hilfsprogramme direkt mittels eines einfachen Texteditors erstellt.

Daher gab und gibt es immer wieder Ansätze, das Potential vorhandener und dem Anwender vertrauter Textverarbeitungssysteme für dieses Ziel zu nutzen und – soweit möglich – weiter auszubauen. Inzwischen bieten viele dieser Programme (wie zum Beispiel *Microsoft Word*, *Adobe FrameMaker* oder *Lotus Word Pro*) dem Benutzer die Möglichkeit, den erstellten Text im HTML-Format zu exportieren. Dabei ist der Autor allerdings auf die Fähigkeiten der HTML-Sprache beschränkt, die sich im jeweiligen Textverarbeitungssystem darstellen lassen. Außerdem läßt leider in vielen Fällen die Qualität der auf diesem Weg erstellten Dokumente noch viel zu wünschen übrig, so daß man manchmal den erstellten HTML-Code noch nachträglich von Hand nachbessern muß.

Ein wesentlicher Vorteil dieses Ansatzes liegt jedoch darin, daß sich das gleiche Dokument sowohl in guter Qualität drucken und als HTML-Version veröffentlichen läßt. Wenn man das Dokument hingegen in einem reinen HTML-Editor erstellt hat, wird man beim Drucken unweigerlich einige unschöne Überraschungen hinsichtlich der Formatierung und Seitenaufteilung erleben.

1.2 Problemstellung

Das Ziel der im Rahmen dieser Diplomarbeit entwickelten Programme ist es, diese Funktionalität zum Erstellen von HTML-Dokumenten für den Editor *Edit* bzw. *TextEdit* zu implementieren, der unter dem Betriebssystem *OpenStep* als Standard-Editor zur Verfügung steht. Dabei soll die Fähigkeit zum Erstellen von HTML-Dokumenten nicht im Editor selbst integriert werden, sondern es wird ein externes Programm zur Verfügung gestellt, daß das Ausgabeformat RTFD (*Rich Text Format Directory*) in HTML umwandeln kann.

RTFD ist eine Erweiterung des ca. 1987 von der Firma *Microsoft* entwickelten Dateiformats RTF (*Rich Text Format*, siehe [RTF 87]) zur portablen und Plattformunabhängigen Speicherung von formatierten Textdokumenten. Es wurde ursprünglich für *Microsoft Word* entworfen, wird aber heute von praktisch jedem Textverarbeitungssystem unterstützt.

Dieses Format wurde von der Firma *NeXT* als Standardformat für ihr *OpenStep* Betriebssystem übernommen und um einige Fähigkeiten erweitert, die gerade als Basis für eine Umwandlung in HTML von besonderer Bedeutung sind:

- Einbettung von beliebigen „Objekten" (Grafik, Audio, Dateien, Kataloge usw.) in den Text
- Hypertext-Links und -Markierungen (Ziele von Hypertext-Links)

  Dies ist nur in *Edit* implementiert, nicht jedoch in *TextEdit*. Für diesen kann man aber eine äquivalente Funktionalität über einen externen *Service* zur Verfügung stellen, siehe dazu Abschnitt 2.2.7 auf Seite 19.

Leider sind diese Erweiterungen nicht genau dokumentiert worden, aber es zeigt sich, daß man durch Ausprobieren leicht die nötigen Informationen sammeln kann.

1.3 Ziele bei der Entwicklung

Das wesentliche Ziel war natürlich die Umwandlung von RTFD-Dokumenten in das HTML-Format, damit diese auch auf nicht *OpenStep*-Systemen gelesen oder im Internet publiziert werden können. Es gab jedoch darüber hinaus noch einige weitere Ziele bei der Entwicklung:

- Das Programm sollte möglichst leicht und einfach zu bedienen sein und möglichst keine Kenntnisse im Bereich RTF oder HTML voraussetzen.
- Es sollte Plattform-unabhängig funktionieren. Damit scheidet eine Lösung mit grafischer Oberfläche schon fast automatisch aus. Sie ließe sich zwar im Prinzip mit *Java* oder *Tcl/Tk* realisieren, doch beide Programmiersprachen haben deutliche Schwächen im Bereich Textbearbeitung.
- Das Programm sollte keine interaktiven Eingaben erfordern, damit es ohne große Probleme von einem Shell-Skript oder Makefile aus aufgerufen werden kann, um eine große Menge von Dateien zu bearbeiten.
- In jedem Fall sollte korrekter HTML-Code erzeugt werden, auch wenn das Eingabedokument niemals dafür vorgesehen war, in HTML umgewandelt zu werden.
- Es sollte möglich sein, ohne (sichtbare) Änderungen im Dokument auszukommen, damit die Druckausgabe immer noch das gewünschte Ergebnis liefert. Trotzdem sollte man, falls notwendig, auch direkt HTML-Code in die Ausgabedatei übernehmen können.
- Das Programm muß sich leicht um neue Funktionen erweitern lassen, ohne dafür den gesamten Programmtext lesen und verstehen zu müssen. Das ist notwendig, da sich die HTML-Sprachdefinition laufend weiterentwickelt und eine schnelle Anpassung erforderlich macht. Die neuen Funktionen sollten sich unabhängig von Rest des Programms testen lassen.
- Eine einfache Navigation zwischen den erzeugten Dokumenten sollte möglich sein, auch wenn diese unabhängig voneinander umgewandelt worden sind. Es wird sich später zeigen, daß dieses Problem schwieriger zu lösen ist, als man vielleicht zunächst vermutet.

Da die wesentliche Aufgabe in der Manipulation von Text besteht, habe ich mich schließlich für die Programmiersprache *Perl* (*Practical Extraction and Report Language*, siehe [Perl 96]) entschieden, die gerade in diesem Bereich durch die direkte Einbindung regulärer Ausdrücke in die Sprache ihre besonderen Stärken besitzt.

Auf die sich aus diesen Entwicklungszielen ergebenden Designüberlegungen werde ich in den nächsten beiden Kapiteln zu den Themen Dokumentarchitektur und Implementierung noch genauer eingehen.

1.4 Vorhandene Grundlagen

Bevor man ein solches Projekt in Angriff nimmt, sollte man sich in jedem Fall umsehen, ob andere Software-Entwickler auf diesem (oder einem verwandten) Gebiet bereits ähnliche Programme entwickelt haben. Vielleicht ist das Problem ja schon von anderen gelöst und das ganze Projekt damit überflüssig geworden. Auch wenn dieser Fall eher unwahrscheinlich ist, so kann man doch häufig auf den Arbeiten anderer Programmierer aufsetzen oder Teile bereits existierender Programme an die eigenen Anforderungen anpassen. Einerseits kann man auf diese Weise zu ganz neuen Ideen und Strategien zur Lösung der eigenen Probleme kommen, andererseits läßt sich viel Arbeit durch die Verwendung bereits getesteter Programme oder Programmfragmente sparen.

Diese Strategie hat sich auch in diesem Fall als sehr hilfreich erwiesen, denn es gab bereits eine Reihe von nützlichen Grundlagen (siehe dazu auf die Quellenübersicht in Anhang A), auf denen die verschiedenen Teile des Umwandlungsprogramms aufbauen:

rtftohtml

Dieses Programm wurde von Chris Hector entworfen, um RTF-Dokumente von verschiedenen Textverarbeitungssystemen für *Windows*- und *Macintosh*-Systeme in HTML umzuwandeln. Auf den ersten Blick erweckt dieses Programm den Eindruck, als könne es die gestellte Aufgabe bereits erfüllen. Leider kommt es mit dem Ausgangsformat RTFD (von *Edit*) nicht besonders gut zurecht. Das hat im wesentlichen drei Ursachen:

- Für die Umwandlung benutzt rtftohtml eine Abbildung von definierten Absatzstilen auf bestimmte HTML-Formatierungsanweisungen, aber leider bietet *Edit* keinerlei Unterstützung für Absatzstile. Außerdem lassen sich viele Ausgabeformate nur über ungewöhnliche Textattribute (wie schattiert, doppelt unterstrichen oder versteckt) einstellen, die in *Edit* ebenfalls nicht zur Verfügung stehen.
- Die Umwandlung der Grafiken muß der Anwender nachträglich von Hand vornehmen, da sich rtftohtml nur um die Umwandlung des Texts kümmert.
- Da das Programm für die Bearbeitung von Rich Text Dateien entwickelt wurde, kann es die Erweiterungen des RTFD-Formats (siehe Abschnitt 1.2 auf Seite 4) nicht erkennen und verarbeiten, obwohl gerade diese für die Umwandlung in HTML wichtig sind.

Dennoch ist das Programm ein nützliches Hilfsmittel für die Umwandlung einfacher Textattribute (fett, kursiv usw.), und es enthält den eigentlichen RTF-Parser, den man sonst selbst implementieren müßte (siehe dazu auch Kapitel 3.2 auf Seite 41).

In RTFD-Dokumenten werden üblicherweise Grafiken in den Formaten TIFF (*Tagged Image File Format*) und EPS (*Encapsulated PostScript*) verwendet, die zur Darstellung in einem Web-Browser zunächst in eines der gebräuchlichen Formate (GIF, JPEG oder PNG) konvertiert werden müssen. Eine Übersicht über die verwendeten Bildformate befindet sich in Anhang B auf Seite 60.

ImageMagick
: ImageMagick ist ein Programmpaket für das *X11 Window System* zur Darstellung und Bearbeitung von Bildern in den verschiedensten Grafikformaten. Es ist besonders gut geeignet für die nicht-interaktive (also programmgesteuerte) Bearbeitung und Umwandlung von Bildern. Daher läßt es sich für die notwendigen Konvertierungen hervorragend einsetzen, auch wenn dabei nur ein kleiner Teil der Funktionalität ausgenutzt wird.

Netpbm
: Netpbm besitzt ähnliche Funktionen wie ImageMagick, aber es gibt keine grafische Oberfläche und das Paket besteht aus einer Vielzahl kleiner Programmodule, die miteinander kombiniert werden können. Dieses Programmpaket ist ausschließlich auf nicht-interaktive Benutzung ausgelegt und hat einen geringeren Funktionsumfang als ImageMagick. Für die erforderlichen Formatkonvertierungen ist es jedoch völlig ausreichend, solange keine transparenten Bilder verwendet werden.

Ghostscript
: Aladdin Ghostscript ist ein Interpreter für Adobe's *PostScript* und *Portable Document Format* Grafik-Beschreibungssprachen. Es kann (unter anderem) Grafiken im EPS-Format darstellen und in viele andere Formate umwandeln. Ghostscript ist ebenfalls gut für nicht-interaktive Benutzung geeignet.

eps2tiff
: Eps2tiff ist ein kleines Hilfsprogramm, das ausschließlich für die Benutzung auf *NeXTStep*/*OpenStep*-System entwickelt wurde und Grafiken vom EPS-Format in das TIFF-Format konvertieren kann. Damit kann es auf diesen Systemen als Ersatz für Ghostscript eingesetzt werden, das im allgemeinen deutlich langsamer ist.

TIFF Utils
: Zu der TIFF Bibliothek von Sam Leffler gehört eine ganze Reihe von kleinen Hilfsprogrammen, die den Umgang mit TIFF-Dateien vereinfachen und einige andere Formate in TIFF umwandeln können. Diese Programme sind hilfreich zur Bearbeitung von Dateien, die mehrere Versionen eines Bilds enthalten (zum Beispiel farbig und schwarz-weiß).

1.5 Gliederung dieser Arbeit

Im Anschluß an diese kurze Einleitung werde ich in dem nun folgenden zweiten Kapitel zunächst das der Umwandlung zugrunde gelegte Dokumentmodell vorstellen und im einzelnen beschreiben, welche Struktur in welcher Form umgewandelt werden kann. Der zweite Teil des Kapitels enthält ausführliche Kommandobeschreibungen aller im Rahmen dieser Diplomarbeit erstellten Programme und stellt anhand kleiner Beispiele vor, wie diese vom Benutzer sinnvoll eingesetzt werden können.

Anschließend folgt dann eine allgemeine Darstellung der Implementierung, in der ich aber nur eine grundlegende Übersicht über das Design und meine dahinter stehenden Überlegungen geben kann. Eine detaillierte Beschreibung der implementierten Programmodule würde jedoch den Rahmen dieser Diplomarbeit sprengen, so daß ich mich im Bereich der eigentlichen Implementierung auf einen relativ kurzen Einblick beschränken muß. Statt dessen werde ich näher auf die von mir gewählte Struktur und Modularisierung der verschiedenen Programmkomponenten eingehen und die Integration der bereits als Grundlage vorhandenen Hilfsprogramme zeigen.

Schließlich werde ich im Kapitel 4 noch einen kurzen Ausblick auf weitere Entwicklungsmöglichkeiten geben und versuchen, die Frage zu beantworten, inwieweit sich der investierte Aufwand letztendlich gelohnt hat, und welche Ziele mit dem von mir verfolgten Ansatz nicht erreicht werden konnten.

Der Anhang enthält noch einige ergänzende Hinweise zu den in dieser Arbeit verwendeten Bildformaten und eine Auflistung der hier bereits erwähnten Hilfsprogramme mit genauen Quellenangaben.

Kapitel 2

Dokumentarchitektur

In diesem Kapitel wird beschrieben, wie man das für diese Diplomarbeit entwickelte Programm *rtfdtohtml* sowie einige dazugehörige kleine Hilfsprogramme einsetzen kann, um eigene Dokumente in die *Hypertext Markup Language* zu übersetzen. Von zentraler Bedeutung ist dabei natürlich zunächst der Aufbau des Ausgangsdokuments und die Art der durchgeführten Umwandlungen.

2.1 Ansatz

Als Ausgangsmaterial für den Umwandlungsvorgang dienen ein oder mehrere Dokumente, die im RTF- oder RTFD-Format vorliegen. Dabei sollte es eigentlich keine Rolle spielen, mit welchem Programm diese Dokumente erstellt worden sind, da das Format ja unabhängig vom jeweils zum Erstellen der Dateien verwendeten Textverarbeitungssystem seien sollte. Leider zeigt sich in der Praxis, daß verschiedene Programme durchaus unterschiedliche Vorstellungen über die Struktur des Texts und damit auch den Aufbau der daraus erzeugten RTF-Quelldatei haben.

Programme wie Adobe's *FrameMaker* oder *Lotus Word Pro* weisen jedem Textabschnitt einen benannten Absatzstil zu, der die einzelnen Eigenschaften des Absatzes genau festlegt. Dazu zählen zum Beispiel die Schriftart und -größe, der Zeilenabstand, Einrücktiefe sowie linker und rechter Randabstand, die Ausrichtung des Texts auf der Seite und die Tabulatorpositionen. Jeder neue Absatz kann entweder den Stil des vorhergehenden übernehmen, oder es ihm kann vom Benutzer ein Stil aus der vorhandenen Liste zugewiesen werden. Selbstverständlich können die vorhandenen Stile auch verändert und neue hinzugefügt werden.

Wird das Dokument nun gespeichert, wird vor dem eigentlichen Text eine Tabelle aller in ihm verwendeten Stile mit den zugehörigen Attributen abgespeichert, das sogenannte *Style Sheet*. Für die Umwandlung in HTML hat diese Struktur einen ganz entscheidenden Vorteil: Für jeden Absatzstil kann eine andere Repräsentierung in Form von HTML-Code gewählt werden, zum Beispiel für verschiedene Arten von Listen, vorformatierten Text, Zitate oder ähnliches; und die Übergänge zwischen den Stilen sind in der RTF-Datei klar gekennzeichnet.

Das eigentliche Problem liegt nun darin, daß der Editor *Edit*, für den das Umwandlungsprogramm entwickelt werden sollte, wie auch sein Nachfolger *TextEdit* einen einfacheren Ansatz verfolgen. Zwar lassen sich auch hier die gewohnten Eigenschaften für einen Textabsatz festlegen, aber es gibt keine benannten Stile, so daß diese Attribute sich nur mit Hilfe von *Copy&Paste* auf einen anderen Absatz übertragen lassen. Einen Stil nachträglich im ganzen Dokument abzuändern ist daher auch praktisch nicht möglich. Außerdem bietet *Edit* keine besonderen Möglichkeiten zur Erstellung von Listen oder Tabellen, beides kann man nur durch die geschickte Anordnung des Texts und der Tabulatorpositionen realisieren.

Diese Einschränkungen sind vielleicht etwas ungewohnt, stellen aber normalerweise für den Benutzer kein großes Problem dar, solange das Dokument nur am Bildschirm betrachtet oder ausgedruckt werden soll. Für die Umwandlung in HTML ergeben sich jedoch aufgrund der fehlenden Kennzeichnung der Stilübergänge in der Ausgangsdatei eine ganze Reihe von Schwierigkeiten, da sich durch die Analyse der Textformatierungen nicht immer eindeutig ein Absatzstil zuordnen läßt. Dieser Umstand führt dann zu manchmal recht merkwürdig wirkenden Kriterien für die Auswahl der zu erzeugenden HTML-Formatierungsanweisungen. Auf viele der damit verbundenen Probleme wird in den folgenden Abschnitten noch genauer eingegangen.

2.2 Dokumentaufbau und Umwandlungen

2.2.1 Text und Überschriften

Das Ausgangsdokument besteht in der Regel (wenn man man einmal eingebette Objekte außer acht läßt) aus einer Folge von Textabsätzen, die bei der Umwandlung jeweils einzeln betrachtet werden. Als Absatz gilt dabei eine nicht leere Zeile, die mit einem Zeilentrenner abgeschlossen ist; der Zeilenumbruch bleibt dabei dem Editor überlassen.

Normaler ASCII-Text wird ohne Veränderung in die Ausgabedatei übernommen, wobei für bestimmte Zeichen die für HTML notwendigen Ersetzungen (> für >, < für < und & für &) vorgenommen werden. Ebenso werden alle Umlaute und Sonderzeichen durch ihre zugehörigen Ersatzdarstellungen ersetzt, sofern diese existieren. Als Sonderzeichen sind dabei alle Zeichen aus den normalen *OpenStep* Schriftarten sowie der *Symbol* Schrift zugelassen. Die Tabelle der vorhandenen Ersatzdarstellungen kann man in [HTML 97] nachlesen.

Da *OpenStep* nicht die bei HTML übliche *ISO-Latin-1* Zeichensatzkodierung verwendet, kann es manchmal vorkommen, daß ein Zeichen nicht direkt in HTML dargestellt werden kann. Dies gilt natürlich erst recht für die *Symbol* Schrift, die fast nur mathematische Symbole enthält. In diesem Fall wird statt dessen eine textuelle Umschreibung (wie `=>` oder `(TM)`) oder einfach der Name des Zeichens (zum Beispiel `delta` an Stelle von δ) in den Text eingefügt, beides kann dabei auch, wie in den Beispielen zu sehen, in der Ausgabe länger als ein Buchstabe sein.

Ein Sonderfall ist die Absatzmarkierung ¶, die in eine horizontale Trennlinie (`<HR>`) umgesetzt wird. Der Grund dafür ist, daß dieses Zeichen als Markierung für einen Seitenumbruch im Dokument beim Drucken über die *Pager*-Applikation[1] verwendet wird. Dieses Programm wurde vor einiger Zeit von Axel T. Schreiner speziell zum Drucken von RTFD-Dateien entwickelt, weil *Edit* praktisch keine Kontrolle über die Positionen der Seitenumbrüche im Dokument ermöglicht. Auf diese Weise erscheinen die einzelnen Seiten auch in einem Web-Browser immer noch optisch voneinander abgesetzt. Eine Absatzmarkierung sollte allein auf einer Zeile stehen.

Die Überschriften sind gegenüber dem normalen Text lediglich durch eine überdurchschnittliche Schriftgröße ausgezeichnet, da sie aus den bereits auf der vorherigen Seite erwähnten Gründen nicht an einem besonders benannten Absatzstil zu erkennen sind. Sie werden häufig auch noch fett gesetzt oder zentriert ausgerichtet, das ist aber für die Erkennung nicht notwendig. Für Überschriften gibt es daher auch keine Sonderbehandlung bei der Umwandlung, es gilt die im nächsten Abschnitt auf Seite 13 beschriebene Zuordnungstabelle für die Schriftgrößen.

Darüber hinaus wird auch die Ausrichtung der Absätze berücksichtigt. Die möglichen Einstellungen hierbei sind linksbündig, rechtsbündig und zentriert. Blocksatz ist weder bei *Edit* noch in HTML vorgesehen. Linksbündig ist wie gewohnt die Voreinstellung, in den anderen Fällen wird der jeweilige Absatz bei der Ausgabe in einen `<DIV align=right>` Block (rechtsbündig) beziehungsweise einen `<DIV align=center>` Block (zentriert) eingebettet. Diese Arten der Ausrichtung funktionieren genauso auch für komplexere Objekte wie Tabellen, Listen und Bilder, auf die später noch im einzelnen eingegangen wird.

2.2.2 Schriftattribute

Es existieren für alle bei *Edit* einstellbaren Schriftattribute entsprechende Formatanweisungen in HTML, viele davon sind jedoch erst in HTML 3.2 neu zum Standard hinzugekommen. Die Abbildung der einzelnen Attribute auf die zugehörigen Formatierungsanweisungen ist in Tabelle 2.1 auf der nächsten Seite zusammengefaßt:

[1] Siehe dazu auch Anhang A auf Seite 59.

Schriftattribut	HTML-Darstellung
fett	<B>
kursiv	<I>
unterstrichen	<U>
kursiv + unterstrichen	<CITE>
durchgestrichen	<STRIKE>[2]
hochgestellt	<SUP>[2]
tiefgestellt	<SUB>[2]

Tabelle 2.1: Zuordnung der Schriftattribute

Alle diese Auszeichnungen sind natürlich auch in Überschriften erlaubt. Beliebige Kombinationen davon sind ebenfalls möglich, wobei dann noch zusätzlich auf die korrekte Verschachtelung der erzeugten HTML-Elemente geachtet wird. Das ist unbedingt notwendig, da zum Beispiel schon folgendes nach [HTML 97] nicht zulässig ist:

```
This has some <B>bold and <I></B>italic text</I>.
```

Für die Darstellung unterschiedlicher Schriftfarben wird der betreffende Textbereich in einen Block der Form <FONT color="#RRGGBB">[2]...</FONT> eingeschlossen, eine Verschachtelung dieser Elemente macht dabei selbstverständlich keinen Sinn, obwohl es nach dem HTML-Standard erlaubt ist. Die voreingestellte Schriftfarbe ist normalerweise Schwarz, sie kann aber auch, wie in Abschnitt 2.3.1 auf Seite 31 gezeigt wird, für die gesamte Datei umgestellt werden.

Änderungen der Schriftgröße können an beliebiger Stelle im Dokument vorkommen und sind unabhängig von den oben beschriebenen anderen Attributen. Die Auszeichnung der Schriftwechsel kann hierbei wahlweise entweder mit den klassischen HTML-Formatanweisungen für Überschriften (<H1> bis <H6>) oder mit den neuen <FONT size=...>[2] Elementen erfolgen. Diese haben gegenüber den <H*n*> Formatanweisungen den Vorteil, daß ein Wechsel der Schriftgröße auch mitten auf einer Zeile möglich ist. Überschriften erzeugen dagegen immer einen Zeilenumbruch bei jedem Schriftwechsel.

Die genaue Zuordnung der Größenbereiche ist in Tabelle 2.2 auf der nächsten Seite aufgeführt. Interessanterweise sind die Überschriftsgrößen 5 und 6 kleiner als der normale Dokumenttext und daher wohl eher für Anmerkungen zu gebrauchen. Für Standardtext ist der Bereich von 9–14 Punkten Schriftgröße vorgesehen, der damit für die meisten Dokumente ausreichend sein sollte (üblich sind etwa 10–12 Punkte). Da jedoch nicht alle Autoren dieselben Schriftgrößen bevorzugen, gibt es außerdem noch die Möglichkeit, alle verwendeten Schriften mit einem frei wählbaren Faktor zu skalieren. Diesen Faktor sollte man gegebenenfalls so wählen, daß der normale

[2] Diese Auszeichnungen sind neu in HTML 3.2.

Text des Dokuments nach der Skalierung in dem angegebenen Bereich von 9–14 Punkten liegt, damit nicht große Teile des Texts als Überschriften erscheinen.

Schriftgröße	HTML-Darstellung
$\geq$ 24 Punkte	Überschrift Größe 1
$<$ 24 Punkte und $\geq$ 19 Punkte	Überschrift Größe 2
$<$ 19 Punkte und $\geq$ 16 Punkte	Überschrift Größe 3
$<$ 16 Punkte und $\geq$ 14 Punkte	Überschrift Größe 4
$<$ 14 Punkte und $\geq$ 9 Punkte	normaler Text
$<$ 9 Punkte und $\geq$ 7 Punkte	Überschrift Größe 5
$<$ 7 Punkte	Überschrift Größe 6

Tabelle 2.2: Zuordnung der Schriftgrößen

2.2.3 Vorformatierter Text

Als *vorformatierter* Text werden hier Textabschnitte bezeichnet, deren Formatierung bereits im Ausgangsdokument festgelegt wird und die bei der Umwandlung in HTML auf jeden Fall beibehalten werden soll. Typische Beispiele dafür sind Auszüge aus Quelltexten und Ausgaben von Programmen, die direkt aus einem Terminalfenster übernommen worden sind. Um solche Bereiche vom übrigen Text abzugrenzen und die Formatierung nicht zu zerstören, werden sie immer in einer nicht-proportionalen Schriftart gesetzt, wie zum Beispiel *Courier* oder *Ohlfs*.

Auch hier wäre eine Erkennung über einen benannten Absatzstil eine ganz einfache Lösung, die aber aus den bereits am Anfang erwähnten Gründen ausscheidet. Daher bietet sich für die Umwandlung nur noch die Möglichkeit an, solche vorformatierten Textbereiche anhand der beiden oben genannten Schriftarten zu erkennen. Dabei sind zwei Fälle zu unterscheiden:

1. einzelne Worte auf einer Zeile

   Die Worte werden bei der Ausgabe in einen `<TT>`...`</TT>` Block eingeschlossen, der auch bei der Betrachtung im einem Web-Browser in einer nicht-proportionalen Schrift dargestellt wird.

2. (mehrere) ganze Zeilen

   Diese werden einfach in der vorhandenen Form komplett in einen großen `<PRE>`...`</PRE>` Bereich eingebettet, der dafür sorgt, daß die Formatierung inklusive aller Leerfelder und Tabulatoren erhalten bleibt. Hierbei muß man darauf achten, daß in einem solchen Bereich keine Änderung der Schriftgröße und keine eingebetteten Objekte (Bilder o.ä.) erlaubt sind.

Leider ergibt sich hier das Problem, daß sich diese Zuordnung nicht immer eindeutig treffen läßt. Unter Umständen reicht schon ein unbeabsichtigtes Leerfeld in der falschen Schrift aus, um die Entscheidung zugunsten von Fall 1 umkippen zu lassen, so daß zum Beispiel ein Stück Quelltext als verschachtelte Liste formatiert wird. Bei Grenzfällen wird daher zusätzlich noch eine Textanalyse durchgeführt, aber trotzdem kann das Resultat anders ausfallen, als es vom Autor gedacht war.

Darüber hinaus gibt es noch einen weiteren Fall, bei dem die Schriftart eines Absatzes eine besondere Bedeutung besitzt:

Analog zum vorformatierten Text ist es auch möglich, direkt HTML-Code aus dem Ausgangsdokument in die Ausgabedatei zu übernehmen, um damit dem Anwender die Möglichkeit offen zu lassen, beliebige Formatierungsanweisungen für eigene Zwecke hinzuzufügen. Auf diese Weise lassen sich zum Beispiel *Java Applets* und *Eingabeformulare* in ein Dokument integrieren. Für diesen Zweck steht eine spezielle, vom Benutzer einstellbare Schrift zur Verfügung, für die keine Umwandlungen durchgeführt werden. Allerdings muß man dabei bedenken, daß diese Texte beim Drucken der RTFD-Datei auch mit im Ausdruck erscheinen.[3]

2.2.4 Tabellen und Listen

Da *Edit* – wie bereits zuvor erwähnt – eigentlich keine Tabellen und Listen kennt, obwohl im RTF-Standard Tabellen vorgesehen sind, muß man diese im Einzelfall durch „geschicktes" Anordnen der Tabulatorpositionen simulieren. Auf diese Weise kann man für Tabellen mehrere Spalten definieren und danach den eigentlichen Text zeilenweise in dieses Grundgerüst eintragen. Bei Listen markieren die Tabulatorpositionen die verschiedenen Einrücktiefen, das funktioniert in gleicher Weise auch für eine mehrfach verschachtelte Listen. Dieser Trick hat aber leider einen gravierenden Nachteil:

Die auf diese Art erzeugten Listen und Tabellen sind nur rein *optisch* als Liste oder Tabelle zu erkennen, da jede Zeile einzeln formatiert wird, obwohl sich zum Beispiel die einzelnen Einträge einer Tabelle durchaus über mehrere Zeilen ausdehnen können. In der RTFD-Datei finden sich jedoch nur unzusammenhängende Textbereiche, die mühsam wieder den einzelnen Spalten zugeordnet werden müssen. Ebenso schwierig kann es sein, den Anfang oder das Ende einer Tabelle zu finden, besonders dann, wenn in die Tabelle Leerzeilen oder Seitenumbrüche eingefügt worden sind.

Daher ist es absolut notwendig, daß der Autor beim Erstellen besonders auf eine saubere Formatierung achtet, denn ein unbedachter oder vielleicht einfach vergessener Tabulator, der sich im Ausgangsdokument gar nicht bemerkbar macht, kann in der erzeugten HTML-Ausgabe leicht die gesamte Formatierung einer Tabelle durcheinanderbringen.

[3]Natürlich besteht immer die Möglichkeit, solche Textabschnitte in der Hintergrundfarbe einzufärben, um sie im Druck zu verstecken.

Als Beispiel betrachte man einmal die folgende kleine Tabelle:[4]

```
                          ▽                   ▽
Dies ist ein sehr langer Text        ⟶| Spalte 2
Spalte 1        ⟶|                   ⟶| Spalte 2
```

Hier ist der erste Eintrag so lang, daß das Tabulatorzeichen bereits zur zweiten Position springt, in der zweiten Zeile sind hingegen zwei Tabs notwendig, um die gewünschte Position zu erreichen. Nach der Umwandlung sieht das dann so aus:

Dies ist ein sehr langer Text	Spalte 2	
Spalte 1		Spalte 2

Das Umwandlungsprogramm hat für die Tabelle eine Breite von drei Spalten ermittelt (hierbei wird der Bereich als ganzes betrachtet) und die Zeilen entsprechend eingetragen, wobei die zweite Zeile in der Mitte ein leeres Feld enthält. Das entspricht jedoch nicht der Absicht des Autors. In diesem Fall läßt sich das Problem ganz einfach durch Löschen der ersten, überflüssigen Tabulatorposition lösen. Um solche Probleme zu vermeiden, sollte man sich an folgende Regeln halten:

- Es müssen immer alle unnötigen Tabulatorpositionen aus dem Bereich für eine Tabelle entfernt werden, auch solche, die hinter dem rechten Rand der Tabelle liegen.
- Bei Tabellen muß jede Zeile mit einem Zeilentrenner abgeschlossen werden, der Text darf nicht am rechten Rand umbrechen.
- Bei Listen sollte man den Text dagegen am rechten Rand umbrechen lassen und für den Absatz den linken Rand entsprechend weit nach rechts einrücken.
- In normalem Text, das heißt außerhalb von Tabellen und Listen, dürfen Tabulatoren nur am Zeilenanfang zum Einrücken verwendet werden, auf keinen Fall mitten auf einer Zeile.

Wenn man bereits etwas Erfahrung im Schreiben von Texten mit *Edit* gesammelt hat, sollte einem dieses keine Probleme bereiten.

Grundsätzlich gibt es bei der Umwandlung in HTML drei verschiedene Möglichkeiten für die Formatierung der Ausgabe:

Tabelle

Dies ist das allgemeinste Ausgabeformat und wird immer dann verwendet, wenn die beiden anderen Formate nicht möglich sind oder sich keine eindeutige Entscheidung treffen läßt. Jedes Tabulatorzeichen in der Eingabezeile wird als Spaltentrenner aufgefaßt und jede Zeile ergibt eine Tabellenzeile in

[4] Die Dreiecke markieren hier die Tabulatorpositionen.

der Ausgabe, dabei muß nicht jede Zeile die volle Anzahl an Spalten besitzen. Aufgrund der eingeschränkten Eingabemöglichkeiten in *Edit* lassen sich keine verschachtelten Tabellen erzeugen.

Definitionsliste

Wenn jede Eingabezeile im betrachteten Bereich mit einem *fett* gesetzten Wort beginnt und danach noch ein Tabulatorzeichen enthält, wird für jede Zeile der gesamte Text dahinter als Erklärung für den Begriff am Zeilenanfang aufgefaßt. Als Begriff gilt hier der ganze Text bis zum Tabulator, das können auch mehrere Worte sein. Die Ausgabe in HTML erfolgt dann in der Form:

```
<DL>
    <DT> Begriff
    <DD> Erklärung
     ...                 (weitere Zeilen)
</DL>
```

Diese Listen können beliebig tief verschachtelt werden, indem man sie durch Tabs oder Verschieben des linken Rands weiter eingerückt. Die anderen Arten von Listen in HTML können nicht generiert werden.

eingerückter Text

Dies ist eigentlich nur ein Spezialfall der Definitionsliste, bei der der Begriffsteil entfällt. Dieses Format wird dann verwendet, wenn ein Bereich nur eingerückt ist und keine Begriffe am Zeilenanfang stehen. Auch hier kann durch entsprechende Verschachtelung jede gewünschte Einrücktiefe erreicht werden.

2.2.5 Eingebettete Objekte

Die im RTF-Standard vorgeschlagene Technik zum Einfügen von Bildern in ein Dokument ist für die Verwendung auf einem *OpenStep*-System leider kaum zu gebrauchen, da sie einerseits zu sehr an *Microsoft Word* orientiert ist (nur BMP, WMF und *Macintosh QuickDraw* Format) und andererseits die Bilder direkt als Binärdaten mit im Dokument abspeichert. Das RTFD-Format bietet dagegen die Möglichkeit, Dateien beliebigen Typs und sogar ganze Kataloge in ein Dokument einzubetten und sie wieder daraus entnehmen zu können, ohne daß die Daten dabei verändert werden. Dies ist möglich, weil sie nicht im Dokument selbst, sondern jeweils als eigene Datei abgelegt werden. Wie das genau organisiert ist, wird später im Abschnitt 3.3 auf Seite 46 vorgestellt.

Bilder in den für RTFD-Dateien üblichen Formaten TIFF und EPS werden bei der Umwandlung automatisch wahlweise in eines der Formate GIF, JPEG oder PNG konvertiert, dieser Vorgang kann allerdings längere Zeit in Anspruch nehmen, wenn das Dokument viele Bilder enthält.

Weiterhin können auch Bilder in den Formaten GIF, JPEG und PNG sowie XBM (*X11 Bitmap*) und XPM (*X11 Pixmap*) direkt in den Text eingefügt werden, sofern für diese Formate unter *OpenStep* ein entsprechendes Grafik-Filterprogramm (zum Beispiel *OmniImageFilter*) installiert ist. Diese Bilder werden dann unverändert in das Ausgabedokument übernommen. Näheres zu den einzelnen Bildformaten kann man in Anhang B auf Seite 60 finden.

Für sonstige Objekte, die nicht als Bilder erkannt worden sind, wird bei der Umwandlung statt dessen ein Hypertext-Link auf das enthaltene Objekt generiert. Man kann so auf einfache Weise Audio- oder Videodateien, Archive, Programme und sogar Kataloge in das Dokument integrieren. Ob der Betrachter damit auch wirklich etwas anfangen kann, hängt dabei natürlich von den Fähigkeiten des verwendeten Web-Browsers sowie dem Vorhandensein eventuell notwendiger *Plug-ins* ab. Hierbei wird noch zusätzlich geprüft, ob diese Objekte auch tatsächlich im Dateisystem vorhanden sind, damit Verweise auf fehlende Dateien auf keinen Fall unbemerkt bleiben.

2.2.6 Hypertext-Links

Der wichtigste Punkt bei der gesamten Umwandlung in HTML ist natürlich die Einbindung von Verweisen auf andere Dokumente oder auch URLs. Diese Verweise können sowohl vollständig (komplette URL mit Protokollangabe und absolutem „Pfad"[5]) als auch relativ zum aktuellen Dokument angegeben werden. Hierbei wird, ebenso wie bei den eingebetteten Objekten, für jeden Verweis auf eine lokale Datei (oder Katalog) geprüft, ob das Ziel auch wirklich im Dateisystem existiert. Lokal bedeutet in diesem Fall, daß die URL keine Protokoll- und Rechnerangabe enthält, also nur aus einem absoluten oder relativen Pfad besteht. Sollte ein solcher Pfad dabei auf eine RTF- oder RTFD-Datei verweisen, wird er nach dieser Prüfung so abgeändert, daß er im Ausgabedokument in jedem Fall auf die dazugehörige HTML-Version zeigt; das heißt, das Programm geht davon aus, daß die Zieldatei ebenfalls umgewandelt wird.[6]

Neben den Hypertext-Links gibt es auch noch Hypertext-Anker, die bei *Edit* als *Marker* bezeichnet werden. Diese dienen als Ansprungpunkte für Verweise innerhalb eines Dokuments. Die Namen für Anker sind eigentlich frei wählbar, dürfen aber nicht die Zeichen `;` und `#` enthalten. Bei der oben erwähnten Prüfung der Links wird allerdings nicht getestet, ob ein eventuell mit angegebener Anker auch im Zieldokument vorhanden ist.

Das Einfügen eines solchen Ankers in *Edit* geschieht über den Menüpunkt *Format/Help/Insert Marker*. Diese sind zunächst im Text nicht sichtbar, was aber mittels *Format/Help/Show Markers* jederzeit umgeschaltet werden kann.

[5] Das muß kein Pfad im Dateisystem sein.

[6] Die Reihenfolge ist dabei ohne Bedeutung.

Für das Einfügen eines Hypertext-Links in das Dokument gibt es verschiedene Möglichkeiten, die jeweils ihre spezifischen Vor- und Nachteile besitzen:

- Ganz analog zu den Ankern kann man in *Edit* über den Menüpunkt *Format/ Help/Insert Link* einen Verweis auf ein anderes Dokument, eine beliebige URL oder auch nur eine andere Stelle in derselben Datei hinzufügen. Diese Verweise sind (wie übrigens auch die *Marker*) eigentlich für das Schreiben von Hilfedateien gedacht, sie funktionieren aber ebenso gut unabhängig davon. Ein besonderer Vorteil ist, daß sie bereits in *Edit* voll funktionsfähig sind (außer für URLs), so daß man die Links direkt testen kann.

  Da sich diesen Verweisen jedoch kein Textbereich zuordnen läßt, ist es nicht möglich, in der bei HTML eigentlich üblichen Weise ein oder mehrere Worte als aktiven Bereich für den Link zu definieren, den man dann im Web-Browser anklicken kann. Statt dessen werden für die Verweise bei der Umwandlung die auch in *Edit* verwendeten blauen Button-Symbole benutzt.

- Um diesen Nachteil zu vermeiden und somit beliebige Teile des Dokuments als aktiven Bereich für einen Link verwenden zu können, gibt es noch eine kleine Variante der oben erwähnten Technik. Wenn man beim Einfügen des Links die *Open File in Edit* Option auswählt, wird ab der aktuellen Position der gesamte Bereich bis zum nächsten *Marker* als aktiver Bereich für diesen Link markiert. Der *Marker* dient dabei nur als Endmarkierung des Texts und erscheint nicht in der Ausgabe, er wird vollständig ignoriert. Außerdem dürfen solche Bereiche auch nicht verschachtelt werden.

  Dies ist zweifellos etwas umständlich in der Verwendung, und es ist auch nur aus dem Grund im Umwandlungsprogramm vorgesehen, damit man die bei HTML üblichen textuellen Links verwenden kann, ohne dadurch das Druckbild des Dokuments zu beeinträchtigen.

- Wenn der Text eines Verweises dagegen ruhig im Dokument sichtbar sein darf, kann man auch eine völlig andere Methode verwenden. Hierbei wird einfach der Textbereich eines Verweises komplett in *blau* eingefärbt, um den Bereich klar abzugrenzen. Zusätzlich muß das Ziel des Links irgendwo in diesem Bereich als Text angegeben werden, und zwar eingeschlossen in die Zeichen < und >. Bei der Umwandlung wird dann dieser Bereich inklusive der beiden Begrenzungszeichen aus dem Text des Dokuments entfernt. Als Nachteil ergibt sich hier aber, daß das Resultat nur in der HTML-Version funktioniert. Das sieht dann zum Beispiel so aus:

  Eingabe: Click here <http://www.uos.de/>!
  Ausgabe: <A href="http://www.uos.de/">Click here !</A>

Selbstverständlich kann man auch alle diese Methoden in einem Dokument mischen, so daß man sich für die jeweils am besten passende entscheiden kann.

2.2.7 Links in TextEdit

Die im vorhergehenden Abschnitt beschriebenen Techniken zur Erstellung von Hypertext-Links in *Edit* funktionieren leider nur sehr eingeschränkt in dessen Nachfolgeprogramm *TextEdit*, denn dieser Editor sieht eigentlich keine Möglichkeit zur Einbindung von Links und Markern in den Text vor. Allerdings betrifft das von den zuvor beschriebenen drei Varianten nur die ersten beiden; die blau eingefärbten Textbereiche funktionieren natürlich unabhängig vom verwendeten Editorprogramm.

Als Ersatz dafür kann man jedoch mit einem vergleichsweise geringen Aufwand eine äquivalente Funktionalität über ein kleines externes Hilfsprogramm implementieren, den *TextLinker*. Wenn dieser auf dem System installiert (und damit als Service registriert) ist, stehen im Menü *Services* bei *TextEdit* zwei neue Menüeinträge zur Verfügung:

- *Services / TextLinker / Edit or Insert Link*

  Dieser Menüpunkt öffnet ein kleines Dialogfenster (dargestellt in Abbildung 2.1), in dem der Benutzer wie gewohnt das Ziel des Links und den Anker im Zieldokument angeben oder später nachträglich editieren kann.

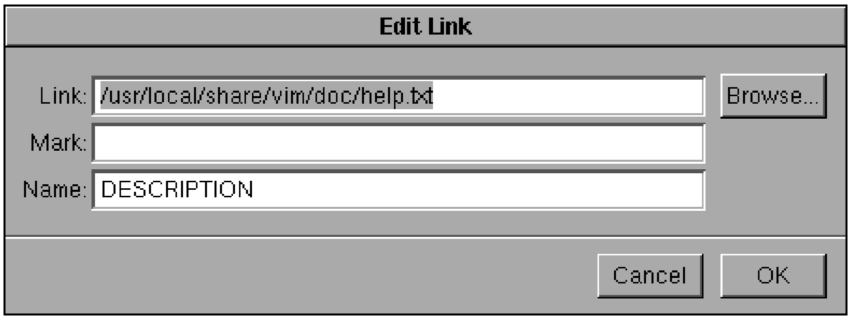

Abbildung 2.1: Dialogfenster des TextLinkers

  Hierbei sollte man nach Möglichkeit für Dateiverweise relative Pfadangaben verwenden, damit das später daraus erzeugte HTML-Dokument nicht mehr von der Position der Quelle im Dateisystem abhängt. Daher trägt das Open-Panel (bei Auswahl über den *Browse*-Knopf) auch nur die jeweils letzte Pfadkomponente der ausgewählten Datei oder des Katalogs in das Eingabefeld ein, eventuell muß diese danach noch entsprechend ergänzt werden.

  Außerdem können in dieses Fenster auch direkt per *Drag&Drop* Links aus einem Web-Browser wie zum Beispiel *OmniWeb* importiert werden. Falls ein solcher Link auf einen Ansprungpunkt im Zieldokument verweist, wird er automatisch zerlegt und entsprechend auf die ersten beiden Eingabefelder verteilt.

- *Services/TextLinker/Edit or Insert Marker*

  Für das Einfügen eines *Markers* wird ein ähnliches Dialogfenster verwendet, in dem jedoch nur der Name des Ankers angegeben bzw. geändert werden kann. Hier macht die Verwendung von *Drag&Drop* natürlich keinen Sinn.

Diese beiden Menüpunkte fügen jeweils die auch schon von *Edit* bekannten blauen bzw. (für die Anker) gelben Button-Symbole an der aktuellen Cursor-Position in den Text ein. Diese können wie gewohnt auch mit *Copy&Paste* normal vervielfältigt werden und sind – nach dem Speichern des Dokuments – bereits im Editor voll funktionsfähig:

Ein Doppelklick auf einen solchen Link öffnet das Ziel des Verweises in der diesem Dateityp über den *Workspace* zugeordneten Applikation (falls es sich um eine lokale Datei handelt) oder in dem eingestellten Web-Browser (bei einer beliebigen URL). Den zum Öffnen einer URL verwendeten Service kann man im *TextLinker* über den Menüpunkt *Info/Preferences...* setzen, voreingestellt ist dort normalerweise der Service-Name `OmniWeb/Open URL`.

Bei der Umwandlung in HTML werden die auf diese Weise erstellten Hypertext-Links und Marker genauso behandelt, wie die im vorherigen Abschnitt beschriebenen äquivalenten Elemente in *Edit*. Es läßt sich allerdings bei der Verwendung des *TextLinkers* leider nicht vermeiden, daß sowohl Links als auch Marker in der Druckausgabe immer sichtbar sind.

2.3 Kommandobeschreibungen

Dieser Abschnitt beschäftigt sich mit den Aufgaben der einzelnen für diese Diplomarbeit entwickelten Programme und ihrer Benutzung durch den Anwender. Außerdem werden hier die Schritte vorgestellt, die notwendig sind, um mehrere Dokumente logisch zu einem Baum zusammenzufassen, in dem der Betrachter sich dann frei bewegen kann.

2.3.1 `rtfdtohtml`

Die zentrale Rolle bei der Konvertierung spielt das Programm *rtfdtohtml*, das den Umwandlungsvorgang startet und kontrolliert. Die zuvor erwähnten Hilfsprogramme (siehe Abschnitt 1.4 auf Seite 6) werden dann bei Bedarf automatisch – und für den Benutzer unsichtbar – aufgerufen. Für den generellen Ablauf muß man sich dabei zunächst zwischen zwei Möglichkeiten entscheiden:[7]

- Die Umwandlung erfolgt normalerweise jeweils im Katalog des Ausgangsdokuments. Auf diese Weise entsteht bei mehreren Dokumenten keine unabhängige Katalogstruktur, sondern die erzeugten HTML-Dateien werden zusammen mit den RTFD-Quellen abgelegt und bilden einen gemeinsamen Dokumentbaum. Dies hat den Vorteil, daß die Information in beiden Formaten immer über denselben Pfad zu erreichen ist. Außerdem müssen bestimmte Dateien, die gemeinsam für beide Formate genutzt werden können, nicht extra kopiert werden. Allerdings muß man von Zeit zu Zeit den Dokumentbaum wieder aufräumen, damit sich keine überflüssigen Dateien ansammeln. Diese Möglichkeit lohnt sich vor allem dann, wenn man dem Betrachter die Dateien in beiden Formaten anbieten möchte.
- Andererseits kann das Programm aber auch für die generierten Ausgabedokumente eine neue und unabhängige Kataloghierarchie erzeugen. Dabei ist dann aber eine gemeinsame Nutzung von Dateien für beide Formate nicht mehr möglich, was unter Umständen zu einem größeren Platzbedarf führen kann. Außerdem muß man hierbei natürlich besonders darauf achten, daß bei relativen Verweisen auf andere Dokumente eventuell auch noch weitere Dateien mit in den Zielkatalog kopiert werden müssen. Die Überprüfung der Links findet in jedem Fall relativ zum Ausgangskatalog statt und kann daher nicht entscheiden, ob im Zielkatalog schon alle notwendigen Dateien vorhanden sind. Diese Möglichkeit bietet sich immer dann an, wenn man ausschließlich an der HTML-Ausgabe interessiert ist.

Wenn man sich für einen Zielkatalog entschieden hat, kann der eigentliche Umwandlungsvorgang gestartet werden.

[7] Die Auswahl erfolgt dabei über die `-output` Option.

Der Aufruf dieses Kommandos erfolgt dann ganz allgemein in der Form:

`rtfdtohtml` [*Option(en)* ...] *Datei(en)* ...

Als Eingabedateien kommen zwei verschiedene Typen in Frage. Es können zum einen normale RTF-Dateien sein, die zwar Hypertext-Links und Anker, aber keine Bilder oder vergleichbare Objekte enthalten können. Die möglichen Dateikennungen hierfür sind *.rtf* sowie *.c*, *.h*, *.m*, *.C* und *.cc* (für Quelltext in C, C++ oder Objective C, der im RTF-Format gespeichert wurde). In jedem Fall wird der Name der Ausgabedatei einfach durch Anhängen der Endung *.html* an den Dateinamen gebildet.

Zum anderen können auch RTFD-Dateien angegeben werden, die im Gegensatz zu RTF auch noch eingebettete Objekte enthalten können. Dieses sind in Wirklichkeit Kataloge mit der Kennung *.rtfd*, in denen die eingefügten Dateien und der RTF-Text getrennt enthalten sind. Die Ausgabedatei trägt in diesem Fall immer den Namen *index.html* und wird direkt in diesem Katalog abgelegt. Zum Beispiel:

```
text.rtf     →   text.rtf.html
test.c       →   test.c.html
datei.rtfd   →   datei.rtfd/index.html
```

Sollte eine angegebene Datei nicht in eine dieser beiden Kategorien fallen, so wird sie bei der Bearbeitung ignoriert. Dadurch kann man in vielen Fällen einfach ein Muster wie `*.rtf*` angeben, ohne daß das Kommando bei der ersten Datei unbekannten Typs abbricht.

Über die folgenden Optionen kann man noch einige Details der Umwandlung beeinflussen und verschiedene Funktionen ein- oder ausschalten:

`-G` oder `--no-graphics`
: Der Umwandlungsvorgang findet intern in zwei getrennten Schritten statt: Zuerst wird der Dokumenttext konvertiert, dann die darin enthaltenen Grafiken. Da sich der Text jedoch im allgemeinen viel häufiger ändert als die in das Dokument eingefügten Bilder, läßt sich über diese Option der zweite Umwandlungsschritt komplett überspringen. Auf diese Weise kann man oft sehr viel Zeit sparen, wenn man weiß, daß wirklich nur der Dokumenttext geändert wurde, denn in den meisten Fällen ist der Aufwand für die Grafikumwandlung deutlich größer als für den Text.

  Die Optionen zur Auswahl der gewünschten Grafikformate müssen allerdings trotzdem mit angegeben werden, damit für die Ausgabe immer noch die richtigen Verweise erzeugt werden können.

`-gif` oder `--gif`
: Diese und die beiden folgenden Optionen bestimmen das Zielformat für die automatische Umwandlung der im Dokument vorhandenen Bilder.

Die Option `-gif` wählt GIF als Umwandlungsformat für die in das Dokument eingefügten EPS-Grafiken. Dieses Format wird auch für die Umwandlung von TIFF-Dateien verwendet, sofern `-jpeg` nicht mit angegeben wird (siehe dort). Wenn keine Option zur Auswahl des Grafikformats angegeben ist, wird als Voreinstellung `-gif` angenommen, so daß in den meisten Fällen die explizite Angabe entfallen kann.

`-png` oder `--png`
: Ganz analog zu `-gif` setzt diese Option PNG als Zielformat für alle im Dokument vorhandenen EPS-Grafiken. Falls `-jpeg` nicht noch zusätzlich angegeben wird, gilt es (wie bei `-gif`) automatisch auch als Zielformat für die Konvertierung von TIFF-Bildern.

  Es kann nicht gleichzeitig GIF und PNG als Format gewählt werden. Sollten dennoch sowohl `-gif` als auch `-png` angegeben worden sein, so bestimmt die letzte dieser Optionen auf der Kommandozeile das Zielformat.

`-jpeg` oder `--jpeg`
: Über diese Option kann das Format für die Umwandlung von TIFF-Dateien festgelegt werden. Wenn `-jpeg` angegeben ist, wird das JPEG-Format verwendet, sonst das für EPS-Grafiken eingestellte Bildformat (GIF oder PNG).

  Wenn nur eine Option (`-gif`, `-jpeg` oder `-png`) angegeben wird, wird also das angegebene Format für alle umgewandelten Bilder (EPS und TIFF) verwendet. Daneben gibt es auch noch die Möglichkeit, verschiedene Zielformate für EPS und TIFF zu verwenden. Da das EPS-Format hauptsächlich für Zeichnungen und Diagramme eingesetzt wird und damit für eine Umwandlung in das Foto-Format JPEG nicht gut geeignet ist und auch die Kombination GIF/PNG nicht sehr sinnvoll ist, gibt es dafür zur Zeit nur die beiden Varianten GIF/JPEG und PNG/JPEG.

  Eine Auflistung aller erlaubten Kombinationen ist noch einmal in der folgenden Tabelle zusammengestellt:

Optionen	Zielformat für EPS	Zielformat für TIFF
`-gif` (oder keine)	GIF (interlaced)	GIF (interlaced)
`-png`	PNG	PNG
`-jpeg`	JPEG (progressive)	JPEG (progressive)
`-gif -jpeg`	GIF (interlaced)	JPEG (progressive)
`-png -jpeg`	PNG	JPEG (progressive)

Tabelle 2.3: Zielformate bei der Grafikumwandlung

Die Beschreibung der Eigenschaften, Vor- und Nachteile der einzelnen Bildformate befindet sich weiter hinten in Anhang B auf Seite 60.

`-head` oder `--heading`

Für die Abbildung der verschiedenen Schriftgrößen in HTML gilt der bereits in Tabelle 2.2 auf Seite 13 dargestellte Zusammenhang. Die Auszeichnung der Schriftwechsel erfolgt dabei normalerweise mit `<FONT size=...>` Elementen, alternativ kann aber auch durch Angabe von `-head` oder `-html2` die Verwendung der entsprechenden HTML-Formatanweisungen für Überschriften (`<H1>` bis `<H6>`) verlangt werden.

`-href` oder `--href`

Diese Option dient zur Auswahl zwischen den auf Seite 18 beschriebenen verschiedenen Techniken zur Erstellung von Hypertext-Links in einem Dokument. Die dort zuerst erwähnte Variante (*Help-Link*) funktioniert immer, die zweite muß dagegen erst durch Angabe der Option `-href` bei der Umwandlung aktiviert werden.

`-blue` oder `--blue`

Ebenso muß die Erkennung der dort beschrieben dritte Variante (blau eingefärbter Textbereich mit Zieladresse in `<...>`) bei der Umwandlung durch diese Option eingeschaltet werden, um eine unbeabsichtigte Erkennung zu vermeiden. Ohne diese Option erscheint der so markierte Bereich einfach als normaler, farbiger Text im Ausgabedokument.

`-html2` oder `--html2`

Aufgrund der stetigen Weiterentwicklung des HTML-Standards durch das *World Wide Web Consortium* gibt es auf Seite der Betrachtungsprogramme inzwischen eine Vielzahl von Programmversionen mit jeweils unterschiedlich umfangreichen Implementierungen. Daher ist es möglich, bei der Umwandlung in HTML eine speziell für die Betrachtung in einem älteren Web-Browser optimierte Version zu erstellen. Dabei werden dann einige Formatanweisungen, die erst ab HTML 3.2 allgemein zur Verfügung stehen, in der Ausgabe entweder durch ältere ersetzt (soweit das möglich und sinnvoll ist) oder einfach ersatzlos gestrichen.

Hierdurch ergeben sich die folgenden Änderungen in den generierten Ausgabedokumenten:

- Jede andere Schriftgröße (außer der für normalen Text verwendeten) wird immer als Überschrift wiedergegeben. Das ist im Ergebnis so, als ob zusätzlich die Option `-head` angegeben worden wäre.
- Bei der Ausrichtung der Absätze erscheinen rechtsbündige Abschnitte wie normaler (linksbündiger) Text. Zentrierter Text wird nicht in einen `<DIV align=center>`, sondern statt dessen in einen `<CENTER>` Block eingeschlossen. Das `<CENTER>` Formatelement war zwar im ursprünglichen HTML 2.0 Standard nicht enthalten, ist aber als Erweiterung schon lange sehr weit verbreitet.

- Der gesamte Text des Dokuments erscheint in Schwarz, da die Darstellung beliebiger Schrift- und Hintergrundfarben in HTML 2.0 noch nicht möglich ist.
- Es werden in der Ausgabe keine Tabellen verwendet. Statt dessen versucht das Programm, die Formatierung soweit möglich allein mit Hilfe von verschachtelten Listen wiederzugeben. Dies kann ja nach Art der Formatierung in der Eingabedatei mehr oder weniger erfolgreich sein, in den meisten Fällen sieht die Darstellung ohne Tabellen jedoch deutlich schlechter aus.

`-index` oder `--index`
: Wenn diese Option angegeben wird, erstellt *rtfdtohtml* noch zusätzlich zu den normalen Ausgabedateien ein sortiertes Stichwortverzeichnis (im folgenden meistens als *Index* bezeichnet) für alle bisher umgewandelten Dokumente. Zum Thema Indexerstellung befindet sich auf Seite 29 noch ein eigener Abschnitt, in dem genau beschrieben wird, welche Wörter in den Index aufgenommen werden und wie dieser aufgebaut ist.

`-no-li` oder `--no-lindent`
: Bei der Berechnung der Einrücktiefe eines bestimmten Absatzes werden sowohl die führenden Tabulatorzeichen (falls vorhanden) als auch die aktuelle Position des linken Rands berücksichtigt. Dabei entspricht jeweils eine Verschiebung des linken Rands nach rechts oder links um mehr als einen vorgegebenen Schwellenwert[8] dem Einfügen beziehungsweise Entfernen eines Tabulators am Zeilenanfang.

  Falls durch dieses Vorgehen zu große Textabschnitte in der Ausgabe eingerückt erscheinen, kann man durch Setzen der Option `-no-li` bei der Umwandlung erreichen, daß alle Änderungen des linken Rands ignoriert werden. Alternativ kann man aber auch probieren, den oben erwähnten Schwellenwert auf einen entsprechend größeren Wert zu setzen.

`-tables` oder `--tables`
: Unter bestimmten Umständen kann es vorkommen, daß sich durch die Verwendung von Tabellen anstelle von Listen im erzeugten HTML-Text eine bessere optische Darstellung erzielen läßt. Daher hat man über diese Option die Möglichkeit, in bestimmten Fällen (zum Beispiel für Definitionslisten) Tabellen gegenüber den Listen in der Ausgabeformatierung zu bevorzugen. Allerdings kann sich hierdurch ebenso leicht eine Verschlechterung ergeben, so daß es sich auf das Ergebnis sowohl positiv als auch negativ auswirken kann. Im Zweifelsfall sollte man zum Vergleich beide Versionen erstellen lassen und dann selbst die bessere auswählen.

[8] Dieser Schwellenwert kann bei Bedarf über die Umgebungsvariable `RTFDLEFTINDENT` eingestellt werden. Siehe dazu Seite 31.

`-nav` *Datei* oder `--nav-file` *Datei*

Als Zusatzfunktion bietet *rtfdtohtml* noch die Fähigkeit zur automatischen Erstellung von Navigationsleisten für die produzierten Ausgabedateien an, die dem Betrachter ein gewisses Maß an Orientierung in der Dokumenthierarchie anbieten und darüber hinaus eine einfache Navigation zwischen mehreren zusammengehörigen Dokumenten erlauben.

Hierfür ist es notwendig, die Beziehungen zwischen den einzelnen Dokumenten (Vorgänger, Nachfolger, Unterkapitel usw.) genau in einer Datei festzuhalten, die dann als Parameter bei der `-nav` Option angegeben werden muß. Auf das Format dieser Datei und die damit verbundenen Einstellmöglichkeiten wird noch genauer im Abschnitt Navigation zwischen Dokumenten auf der nächsten Seite eingegangen.

`-output` *Katalog* oder `--output` *Katalog*

Normalerweise werden die angegebenen Dokumente sozusagen *„in-place“* bearbeitet, damit ist gemeint, daß die Ausgabedateien jeweils in den Katalog des Ausgangsdokuments geschrieben werden. Alternativ erlaubt diese Option dem Benutzer, einen Pfad vorzugeben, unter dem alle im Laufe des Umwandlungsvorgangs erzeugten Dateien (inklusive der konvertierten bzw. nur kopierten Bilder) abgelegt werden sollen. Dabei wird die beim Aufruf angegebene Katalogstruktur an der Zielposition vollständig wieder aufgebaut:

```
$ rtfdtohtml -output /tmp Buch/eins.rtf Buch/zwei.rtf
rtfdtohtml: Processing Buch/eins.rtf...
rtfdtohtml: Processing Buch/zwei.rtf...
$ ls -a /tmp/Buch
.    .!link.gif    ..    eins.rtf.html    zwei.rtf.html
```

Hier ist der Katalog `/tmp/Buch` automatisch angelegt worden. Die jeweiligen Vor- und Nachteile dieser beiden Vorgehensweisen sind bereits am Anfang dieses Abschnitts auf Seite 21 ausführlich diskutiert worden.

`-scale` *Faktor* oder `--scale` *Faktor*

Um die von verschiedenen Autoren verwendete unterschiedliche Basisschriftgröße (die Schriftgröße der gewöhnlichen Textabsätze) auszugleichen, ist für die Abbildung der verwendeten Größen auf die HTML-Formatelemente keine feste Zuordnung vorgegeben. Statt dessen kann man über die Option `-scale` sämtliche Schriftgrößen bei der Umwandlung noch mit einem geeigneten Skalierungsfaktor versehen, bevor die in Tabelle 2.2 auf Seite 13 dargestellte Zuordnung stattfindet.

Falls notwendig, sollte der Umrechnungsfaktor so gewählt werden, daß die Basisschriftgröße nach der Skalierung im dort angegebenen Bereich von 9–14 Punkten liegt. Die übrigen Größen werden dann ebenfalls entsprechend angepaßt. Allerdings gilt diese Option *ausschließlich* für den Text des Dokuments und beeinflußt in keiner Weise die Umwandlung von eventuell eingefügten Bildern.

`-version` oder `--version`
Diese Option gibt lediglich die aktuelle Versionsnummer von *rtfdtohtml* sowie Copyright-Informationen auf die Standardausgabe aus und hat sonst keine Bedeutung. Unmittelbar danach wird das Programm beendet.

Navigation zwischen Dokumenten

Der bisher betrachtete Fall, daß ein einzelnes oder mehrere unabhängige Dokumente umgewandelt werden sollen, tritt in der Realität eigentlich eher selten auf. Typischerweise bestehen größere Projekte – wie zum Beispiel Dokumentationen, Artikel oder Skripte – aus vielen kleinen Dateien, die bei der Umwandlung nicht isoliert für sich betrachtet werden können. Es wäre doch für den Betrachter sehr unpraktisch, wenn er nur mit viel Aufwand oder überhaupt nicht zum nächsten Kapitel gelangen könnte.

Daher war bereits eines der von Anfang an vorgegebenen Designziele, daß sich der Umwandlungsvorgang nicht nur auf die Bearbeitung von einzelnen Dateien beschränken, sondern auch die Beziehungen zwischen den Dokumenten dabei berücksichtigen sollte. Im Ergebnis muß sich der Betrachter auf möglichst angenehme Weise zwischen den verschiedenen Teilen einer größeren Dokumentensammlung hin und her bewegen können. Dieses läßt sich am einfachsten mittels einer zusätzlichen Navigationsleiste innerhalb der Dateien erreichen, die den Zugang zu logisch verwandten Dokumenten ermöglicht, zum Beispiel zum nächsten oder vorhergehenden Kapitel, zum Inhalts- oder Stichwortverzeichnis.

Allerdings ergibt sich bereits bei der Beschaffung der vollständigen Dateiliste ein nicht zu unterschätzendes Problem:

Da bei einem Aufruf von *rtfdtohtml* üblicherweise nicht alle, sondern nur ein Teil der vorhandenen Dokumentensammlung angegeben wird und damit die Argumentliste nicht sehr aussagekräftig ist, muß sich das Programm auf andere Weise die Informationen über alle zum Projekt gehörigen Dateien und deren logische Verbindungen untereinander beschaffen. Dazu sind generell viele und auch beliebig komplizierte Möglichkeiten denkbar, ich habe mich im hier vorliegenden Fall für einen noch relativ einfachen Weg entschieden, um den dafür notwendigen Zusatzaufwand relativ gering zu halten.

Als Grundlage für die automatische Erstellung der Navigationsleisten muß die logische Struktur der gesamten Dokumenthierarchie an zentraler Stelle[9] in einer einfachen Textdatei beschrieben sein. Das genaue Format dieser Datei möchte ich anhand des nachfolgenden Beispiels näher erläutern:

[9] Damit ist der oberste Katalog der betrachteten Dokumenthierarchie gemeint.

```
# Dies ist ein einfaches Beispiel

cont-string:  Inhalt
next-string:  Vorw&auml;rts
prev-string:  Zur&uuml;ck
mail-string:  <IMG src="email.gif">
mailto:       user@foo.bar

Inhalt.rtf
"Kapitel 1.rtf"         title="Einführung"
"Kapitel 2.rtf"         title="Benutzung"
{
    "Abschnitt 2.1.rtfd"
    "Abschnitt 2.2.rtf"
    code/Beispiele.html
}
"Kapitel 3.rtfd"        title="Ausblick"
Anhang.rtf
```

Beispiel 2.2: Aufbau einer Beschreibungsdatei

Leere Zeilen und solche, die mit einem `#` beginnen (Zeile 1 im Beispiel), gelten als Kommentarzeilen und werden beim Einlesen ignoriert. Abgesehen davon gibt es ganz allgemein nur zwei Arten von Zeilen:

Schlüsselwort-Definition

Eine solche Definition hat immer die Form *Schlüsselwort*: *Text* und wird verwendet, um die Werte für eine ganze Reihe von Variablen zu setzen. Dabei muß der angegebene Doppelpunkt unmittelbar (ohne Zwischenraum) auf das Schlüsselwort folgen, als Wert gilt dann der gesamte Text dahinter bis zum Zeilenende. Die möglichen Schlüsselworte sind in der folgenden Tabelle aufgelistet, die Voreinstellung ist jeweils in Klammern angegeben:

Schlüsselwort	Bedeutung
`prev-string`	Text für die Schaltfläche *Zurück* (Previous)
`next-string`	Text für die Schaltfläche *Vorwärts* (Next)
`up-string`	Text für die Schaltfläche *Nach Oben* (Up)
`cont-string`	Text für die Schaltfläche *Inhalt* (Contents)
`index-string`	Text für die Schaltfläche *Index* (Index)
`mail-string`	Text für die Schaltfläche *E-Mail* (Feedback)
`index`	optional: Name oder URL der Index-Datei
`mailto`	optional: E-Mailadresse des Autors

Tabelle 2.4: Liste der Navigationsvariablen

Alle für diese Variablen angegebenen Werte werden unbearbeitet in die Ausgabedatei übernommen, so daß man auch beliebige HTML-Formatelemente darin verwenden kann, im Beispiel wird auf diese Weise ein Bild anstelle eines Texts eingesetzt. Das bedeutet aber auch, daß man alle Zeichen, die in HTML eine Sonderbedeutung besitzen (insbesondere `<`, `>` und `&`) sowie Umlaute und Sonderzeichen stets durch ihre entsprechenden Ersatzdarstellungen ersetzen muß, also zum Beispiel wie in Zeile 4 `Vorwärts` anstelle von `Vorwärts`.

Dokumenteintrag

Die übrigen Zeilen sind die einzelnen Dokumenteinträge in der gewünschten logischen Anordnung. Das erste Dokument sollte (falls vorhanden) stets das Inhaltsverzeichnis sein, daran anschließend folgen der Reihe nach die weiteren Dokumente, wobei Unterkapitel – wie im Beispiel zu sehen – durch Bildung einer Blockstruktur gekennzeichnet werden. Innerhalb eines solchen Blocks verweist *Nach Oben* immer auf das Dokument direkt vor dem jeweiligen Block, es können sogar mehrere Blöcke zu einem Dokument gehören, so daß sich eine Baumstruktur ergibt. Außerdem läßt sich das ganze (theoretisch) beliebig tief verschachteln.

Eine einzelne Zeile enthält den Dateinamen des Dokuments, der exakt so angegeben werden muß, wie später auf der Kommandozeile von *rtfdtohtml*, eventuell inklusive führender Katalogbezeichnungen. Dahinter kann noch mit der Anweisung `title="`*Name*`"` der Titel für dieses Dokument festgelegt werden. Wenn diese Angabe fehlt, wird einfach der Dateiname ohne die Kennung verwendet. Alternativ kann statt eines Dateinamens auch eine beliebige URL angegeben werden, allerdings fehlt dort im allgemeinen die auf diese Weise erzeugte Navigationsleiste.

Man erkennt im Beispiel auch, daß ein ein Dateiname immer in doppelte Anführungszeichen eingeschlossen werden muß, wenn er Leerfelder, Doppelpunkte oder sonstige Zeichen mit Sonderbedeutung enthält. Innerhalb der Anführungszeichen werden dann noch \" und \\ durch " bzw. \ ersetzt.

Automatische Indexerstellung

Ein weiterer interessanter Punkt beim Umwandlungsvorgang ist die Möglichkeit, vollautomatisch ein sortiertes Stichwortverzeichnis für alle bearbeitete Dokumente erstellen zu lassen, das die gesammelten Querverweise in Form von Hypertext-Links auf die dazugehörigen Dateien zur Verfügung stellt. Auch wenn sich damit natürlich kein vollwertiges Textindizierungssystem wie zum Beispiel *Glimpse* ([Manber 94]) ersetzen läßt, so erreicht man doch schon einen beachtlichen Nutzen ganz ohne zusätzlichen Aufwand für den Autor des Texts.

Hier ergibt sich aber wieder einmal das schon bekannte Problem, daß zwar der RTF-Standard die Markierung von Textstellen als Indexeintrag vorsieht, dies aber in *Edit* nicht implementiert ist und daher auch nicht verwendet werden kann. Also müssen die Stichwörter für den Index auf andere Weise im Text aufgefunden und zusammengestellt werden. In das Stichwortverzeichnis werden aufgenommen:

- Überschriften

  Als Überschrift gilt hierbei alles, was in einer größeren Schrift als der Basisschriftgröße des Dokuments (die Schriftgröße der gewöhnlichen Textabsätze) formatiert ist.

- Begriffsdefinitionen

  Damit sind die in Abschnitt 2.2.4 auf Seite 16 genauer beschriebenen Definitionslisten gemeint.

- kursiv gesetzte Textbereiche

  Ein zusammenhängender Bereich aus einem oder mehreren Worten, der mit einem Großbuchstaben beginnt und keine Sonderzeichen enthält. Die Umlaute, Ziffern und bestimmte Satzzeichen (Klammern, Bindestrich, Komma, Semikolon) gelten dabei nicht als Sonderzeichen.

Es wird jedoch dasselbe Stichwort pro Datei nur höchstens einmal erkannt und als Indexeintrag markiert, auch wenn es mehrfach in der Datei vorkommt. Wenn ein Stichwort dagegen in mehreren Dateien auftritt, erscheint es auch entsprechend häufig (einmal für jede Datei) im Index.

Da der Index eigentlich keinem speziellen Dokument zugerechnet werden kann und normalerweise (wie bereits im letzten Abschnitt erwähnt) niemals alle Dokumente bei einem Aufrufe von *rtfdtohtml* angegeben werden, tritt hier nun das Problem auf, daß Informationen über mehrere Programmaufrufe hinweg gesammelt werden müssen. Dieses Problem läßt sich aber noch recht leicht dadurch lösen, daß man einen Index-Cache einführt, der bei jedem neuen Umwandlungsvorgang um die neuen Einträge ergänzt und aktualisiert wird. Leider ergibt sich durch diese Lösung der Nachteil, daß dieser Cache nicht immer automatisch aktualisiert werden kann. Wenn zum Beispiel ein Dokument gelöscht worden ist, bleiben die auf diese Datei bezogenen Indexeinträge zunächst erhalten. In diesem Fall kann es daher notwendig werden, den Index-Cache von Hand zu löschen (er befindet sich in der Datei *.index.in* im Zielkatalog), damit er beim nächsten Programmaufruf neu aufgebaut wird.

Falls die Indexerstellung mit der im vorherigen Abschnitt behandelten Erzeugung von Navigationsleisten kombiniert wird, bekommt auch die Index-Datei diese Navigationsmöglichkeit, und der Index erscheint automatisch in der Navigationsleiste aller umgewandelten Dokumente.

Umgebungsvariablen

Es gibt noch einige weitere Einstellmöglichkeiten, die nur über Umgebungsvariablen beeinflußt werden können, weil sie normalerweise nur selten geändert und typischerweise für eine große Zahl von Dokumenten immer gleich gesetzt werden. Dies könnten im Prinzip auch normale Optionen sein, doch dann müßten sie bei jedem Aufruf von *rtfdtohtml* immer mit angegeben werden, was jedoch nur unnötig aufwendig wäre.

`RTFDBODYATTR`
: Der Wert dieser Umgebungsvariablen gibt an, welche Attribute in der HTML-Ausgabe für das `<BODY>` Formatelement verwendet werden sollen. Die Voreinstellung ist `bgcolor=White` für einen weißen Hintergrund. Wenn man zum Beispiel rote Schrift auf weißem Hintergrund einstellen möchte, kann man das folgendermaßen erreichen:

```
RTFDBODYATTR="text=Red bgcolor=White"
export RTFDBODYATTR
```

Die möglichen Attribute sind `bgcolor` für die Hintergrundfarbe, `text`, `link`, `vlink` und `alink` für verschiedene Textfarben und `background` für ein Hintergrundbild, siehe dazu auch [HTML 97].

`RTFDGAMMA`
: Der hierfür angegebene Wert legt den Faktor für die Gammakorrektur bei der Grafikumwandlung fest. Die Voreinstellung ist 1.0 und braucht normalerweise nicht geändert zu werden. Allgemeine Informationen zur Verwendung der Gammakorrektur bei Bildern kann man zum Beispiel bei [PNG 96] nachlesen.

`RTFDLEFTINDENT`
: Damit das Programm nicht zu empfindlich auf Änderungen der linken Randeinstellung reagiert, gibt es einen Schwellenwert, bis zum dem eine Verschiebung des linken Rands bei der Umwandlung ignoriert wird (siehe Seite 25). Wenn über diese Variable nichts anderes eingestellt worden ist, wird der Wert 75 verwendet. Die Angabe erfolgt dabei in Punkten (ein Punkt ist $^1/_{72}$ Inch).

`RTFDLIBDIR`
: Diese Umgebungsvariable enthält den Pfad zu dem Katalog, in dem die verschiedenen Konfigurationsdateien von *rtfdtohtml* abgelegt sind. Er wird bei der Installation festgelegt und ist meistens `/usr/local/lib/rtfdtohtml`. Diese Variable umzusetzen ist nur dann sinnvoll, wenn man die Konfigurationsdateien kopieren und für eigene Zwecke anpassen möchte.

`RTFDQUOTEFONT`
: Wie bereits in Abschnitt 2.2.3 auf Seite 14 beschrieben, besteht die Möglichkeit, direkt HTML-Code aus dem Ausgangsdokument in die Ausgabedatei

übernehmen zu lassen. Dieser Text muß dazu in einer speziellen Schrift formatiert werden, die mittels dieser Variablen eingestellt werden kann. Die Einstellmöglichkeit ist erforderlich, weil natürlich nicht überall dieselben Schriften zur Verfügung stehen. Die für diesen Zweck voreingestellte Schriftart ist *Tekton*.

2.3.2 `rmrtfd`

Falls die Umwandlung von *rtfdtohtml* im Katalog des Ausgangsdokuments durchgeführt worden ist (wie auf Seite 21 beschrieben), man aber dennoch die als Ergebnis der Umwandlung entstandenen HTML-Seiten von den Quelldateien trennen möchte, kann man mit diesem Kommando die Ausgangsdateien für die auf der Kommandozeile angegebenen RTF- oder RTFD-Dateien entfernen. Anschließend ist nur noch die HTML-Version der angegebenen Dokumente vorhanden.

rmrtfd löscht dabei alle Dateien, die für die Betrachtung der generierten HTML-Seiten nicht mehr erforderlich sind, also insbesondere auch die TIFF- und EPS-Bilder sowie die RTF- und RTFD-Quelldateien, für die diese Umwandlung durchgeführt wurde. Daher sollte man hier sorgfältig darauf achten, daß man nicht versehentlich Dateien löscht, die noch benötigt werden, so daß man dieses Kommando im allgemeinen wohl nur auf eine Kopie des Originalkatalogs anwenden sollte.

Selbst wenn als Ziel für die Umwandlung ein getrennter Katalog gewählt wurde, kann der Aufruf von *rmrtfd* in dem entsprechenden Zielkatalog dennoch sinnvoll sein, um die dort von allen Dokumenten gemeinsam benutzten Dateien (wie zum Beispiel Bilder oder Animationen) in einem zentralen Katalog zu sammeln. Siehe dazu die Beschreibung der `-shared` Option weiter unten auf dieser Seite. Natürlich können in diesem speziellen Fall keine Quelldateien gelöscht werden, da diese im Zielkatalog ja nicht vorhanden sind.

Der Aufruf dieses Kommandos erfolgt wie üblich in der Form:

`rmrtfd` [*Option(en)* …] *Datei(en)* …

Es gibt zur Zeit die folgenden Optionen:

`-shared` oder `--shared`
: Falls in den Ausgangsdokumenten bestimmte Objekte (wie Symbole oder Trennlinien) häufiger verwendet werden, kann es sehr sinnvoll sein, die dazugehörigen Dateien nur einmal in einem von allen Dokumenten gemeinsam benutzten Katalog abzulegen. Mit dieser Option läßt sich vermeiden, daß mehrfach auftretende Dateien gleichen Inhalts an mehreren Stellen gespeichert und damit eventuell später auch mehrfach übertragen werden müssen.

Dabei wird zunächst jeder Datei ein eindeutiger Name zugewiesen, anschließend werden für alle auf der Kommandozeile angegebenen Quelldateien die Verweise in den daraus erzeugten HTML-Dateien entsprechend korrigiert.

Sollten in dem gemeinsam benutzten Katalog schon Dateien vorhanden sein, werden diese bei der Betrachtung bereits berücksichtigt, und es werden nur neue, von diesen verschiedene Dateien hinzugefügt.

`-link` oder `--link`
: Diese Option ist nur in Verbindung mit `-shared` sinnvoll und bewirkt zusätzlich, daß alle gemeinsam benutzten Dateien in den zentralen Katalog gelinkt werden, anstatt sie dorthin zu verschieben. Das hat den Vorteil, daß man (ohne zusätzlichen Platz auf der Festplatte zu verbrauchen) stets am *Link-Count* der Dateien erkennen kann, aus wievielen HTML-Dateien jeweils auf sie verwiesen wird.

Über die Umgebungsvariable `RTFDSHARED` kann schließlich noch der Name für den gemeinsam benutzten Katalog festgelegt werden. Wenn nichts anderes angegeben wird, bekommt er den Namen `shared`, der auf den Verwendungszweck hindeuten soll.

2.3.3 `rmhtml`

Auch dieses Kommando ist für den Fall vorgesehen, daß die Umwandlung an der Position des Ausgangsdokuments durchgeführt worden ist (ohne Verwendung der `-output` Option) und man einen Teil des Dokumentbaums aufräumen möchte. Es arbeitet ganz analog zu dem oben beschriebenen *rmrtfd*, dient jedoch zum Löschen der aus den auf der Kommandozeile angegebenen RTF- oder RTFD-Dateien erzeugten HTML-Dateien und der dazugehörigen umgewandelten Bilder.

Daher wird es auch in der gleichen Weise aufgerufen, besitzt allerdings in Gegensatz zu *rmrtfd* keine Optionen:

> `rmhtml` *Datei(en)* ...

Wenn bei der Umwandlung die Option `-index` angegebenen wurde, muß man jedoch folgendes beachten: Die Datei *index.rtf.html*, in der das dokumentübergreifende Stichwortverzeichnis gespeichert ist, sowie der zuvor beschriebene Index-Cache *.index.in* werden hierbei nicht mit gelöscht, da diese Dateien keinem Ausgangsdokument direkt zugeordnet werden können. Diese müssen dann bei Bedarf nachträglich von Hand gelöscht werden.

2.3.4 `compress_gifs`

Für das GIF-Format wird normalerweise als Komprimierungsverfahren der LZW-Algorithmus (benannt nach *Lempel-Ziv & Welch*) verwendet, der jedoch einem Patent der Firma *Unisys* unterliegt und daher nicht ganz uneingeschränkt verwendet werden kann. Dieser Umstand hat dazu geführt, daß es Versionen von *ImageMagick* mit und ohne die Möglichkeit zur LZW-Komprimierung unter anderem von GIF-Dateien gibt. Wenn man nur die Version ohne diese Komprimierung zur Verfügung hat, kann man mit dem Kommando `compress_gifs` die bei der Umwandlung erzeugten Dateien noch nachträglich komprimieren lassen. Der Aufruf erfolgt einfach mit:

> `compress_gifs` *Datei(en)* ...

Dieses Kommando benutzt dazu die beiden Programme *giftopnm* und *ppmtogif* aus Jef Poskanzers *Netpbm* Paket, das eine große Zahl von derartigen kleinen Bausteinen zur Bearbeitung und Umwandlung von Bildern enthält. Die Verwendung ist natürlich nur für Dateien sinnvoll, die aus RTFD (nicht RTF) übersetzt wurden, da nur diese überhaupt Bilder enthalten können. Andere Dateien dürfen zwar mit angeben werden, sie werden dann aber einfach übergangen.

Man sollte dieses Programm eigentlich nur als Notlösung einsetzen, wenn es aus irgendeinem Grund nicht möglich ist, entweder die LZW-Version von *ImageMagick* zu installieren oder anstelle von GIF das PNG- oder das JPEG-Format zu benutzen.

Kapitel 3

Implementierung

Die zu lösende Aufgabe besteht im wesentlichen aus zwei Teilbereichen, die voneinander unabhängig betrachtet werden müssen:

- die Umwandlung des Dokumenttexts in den entsprechenden HTML-Code
- die Umwandlung der Bilder in ein passendes, für einen Web-Browser geeignetes Grafikformat

Daher werde ich zunächst noch einmal kurz auf einige Designüberlegungen und die grundlegende Struktur eingehen sowie die sich daraus ergebenden Konsequenzen beschreiben und anschließend in zwei getrennten Abschnitten die Implementierung der Text- beziehungsweise Grafikumwandlung darstellen. Dort werde ich dann auch genau auf die Probleme eingehen, die sich aus der Integration der verschiedenen Hilfsprogramme ergeben haben. Abschließend folgt noch ein kurzer Abschnitt zur Implementierung des *TextLinkers*, in dem ich allerdings eingehende Kenntnisse über die Programmierung im Bereich *OpenStep* voraussetzen muß.

3.1 Struktur und Modularisierung

Wie ich bereits bei der Darstellung der Entwicklungsziele im Abschnitt 1.3 auf Seite 4 angemerkt habe, war eines der wesentlichen Ziele die leichte und einfache Erweiterbarkeit des Programms, um es schnell an neue Anforderungen anpassen zu können. Dabei sollte man sinnvollerweise auch berücksichtigen, daß sich die neu implementierten Funktionen auch möglichst einfach, vielleicht sogar vollständig unabhängig vom Rest des Programms, testen lassen sollten. Um diese Anforderungen optimal erfüllen zu können, habe ich eine ziemlich ungewöhnliche Programmstruktur gewählt:

Das vom Benutzer aufgerufene Kommando *rtfdtohtml* ist in Wirklichkeit nur ein relativ kurzes Shell-Skript, das sich als Hauptaufgabe mit der Abarbeitung der Kommandozeilenoptionen und Umgebungsvariablen beschäftigt. Der eigentliche Umwandlungsvorgang (sowohl im Text- wie auch im Grafikbereich) wird dann von einer großen Zahl von kleinen, voneinander (soweit wie möglich) unabhängigen Programmodulen erledigt, die jeweils ihre Standardeingabe entsprechend der ihnen zugeordneten Aufgabe bearbeiten und das Resultat wieder auf der Standardausgabe ausgeben. Diese kleinen Bausteine werden von *rtfdtohtml* in einer langen Kette von Pipelines einfach so hintereinandergeschaltet, daß sich insgesamt die gewünschte Funktionalität ergibt.

Je nach der Art der Eingabedatei (RTF, RTFD oder Quelltext) und den auf der Kommandozeile angegebenen Optionen werden unter Umständen bestimmte Module aus der Pipeline weggelassen oder weitere hinzugefügt. In diesem Konzept übernimmt *rtfdtohtml* dann nur noch die Ablaufsteuerung des Umwandlungsvorgangs. Im Bereich der Grafikumwandlung wurde diese Ablaufsteuerung aus Gründen der Übersichtlichkeit in ein eigenes Shell-Skript ausgelagert (*anytoimg.sh*, siehe dazu Abschnitt 3.3 auf Seite 45), das je nach Ausgangs- und Zielformat einen der zur Verfügung stehenden Grafikfilter startet.

Dieser Aufbau bietet den Vorteil, daß jedes einzelne Modul ganz einfach und interaktiv direkt von der Kommandozeile aus getestet werden kann. Das gleiche gilt natürlich auch für das Testen eines neuen Moduls, es kann vor dem Einsatz schon mit echten Eingabedateien auf eventuelle Schwachstellen untersucht werden. Ebenso ist weitgehend sichergestellt, daß der Einbau eines neuen Moduls die bereits vorhandene Funktionalität nicht beeinträchtigt, da der existente Programmtext (mit Ausnahme von *rtfdtohtml*) dazu nicht verändert werden muß. Aus diesem Grund ist es auch nicht notwendig, sich in die Funktionsweise der anderen Module einzuarbeiten, wenn man das Programm nur um ein neues erweitern möchte. Ein direkter Eingriff in ein bestehendes Modul birgt dagegen immer die Gefahr, neue Fehlerquellen einzuschleppen.

Weil das hier gewählte Design zudem eine Kommandozeilen-orientierte Implementierung geradezu nahelegt, läßt sich auch ein weiteres wichtiges Entwicklungsziel leicht erreichen:

Das Programm kann ohne Probleme nicht-interaktiv betrieben werden und ist damit ohne weiteres für einen Einsatz in anderen Shell-Skripten oder in Makefiles geeignet. Auf diese Weise können dann große Mengen von Eingabedateien im Hintergrund bearbeitet werden, ohne daß eine Kontrolle durch den Benutzer erforderlich ist.

Um die bisher beschriebene grobe Struktur zu verdeutlichen, habe ich sie noch einmal in der Ablaufskizze auf der nächsten Seite zusammengefaßt:

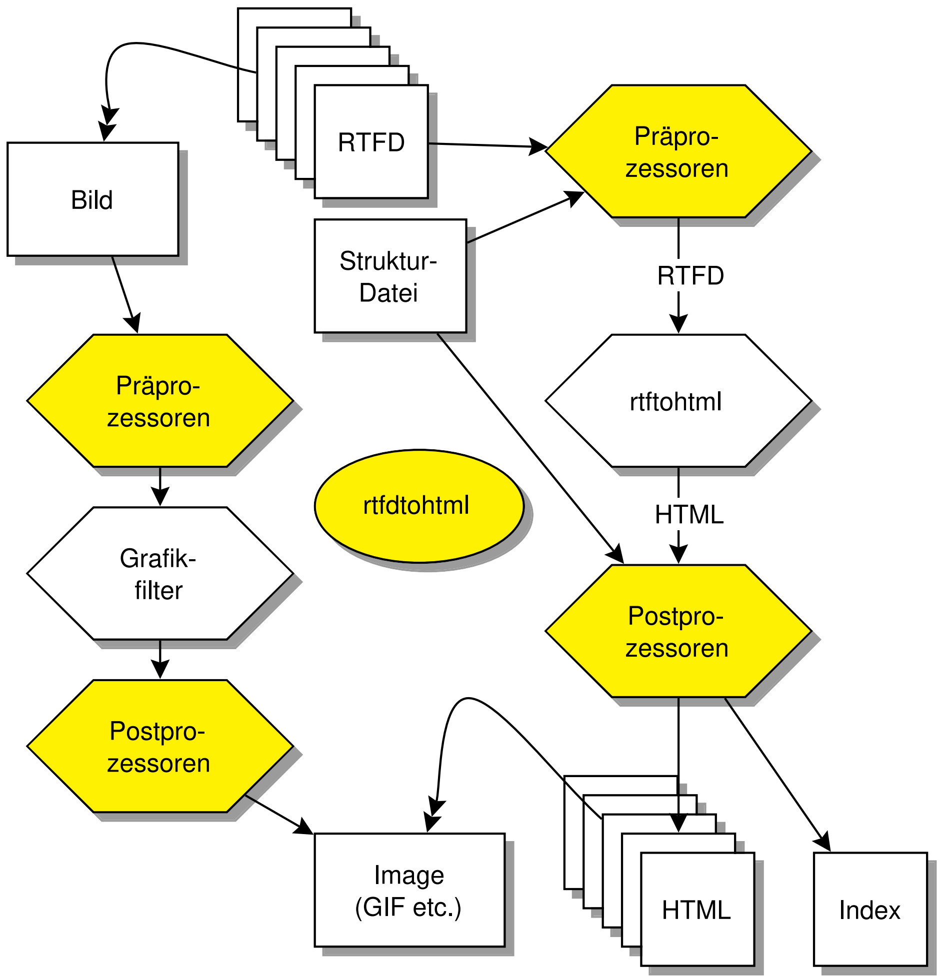

Abbildung 3.1: Ablaufskizze des Umwandlungsvorgangs

Da die zu lösende Aufgabe im wesentlichen in der Manipulation von Textbausteinen bestand, habe ich fast alle Module (bis auf *preproc.sed* und die Grafikumwandlung) in der Programmiersprache *Perl*[1] implementiert, da es durch die direkte Einbindung von Mustererkennung mit regulären Ausdrücken in die Sprache für diesen Zweck besonders gut geeignet ist und eine an C angelehnte Syntax besitzt.

Natürlich könnte man darauf aufbauend auch noch zusätzlich eine grafische Oberfläche implementieren. Da das Programm aber möglichst Plattform-unabhängig funktionieren sollte, habe ich hierauf zunächst einmal verzichtet. Einige weitere Überlegungen zu diesem speziellen Aspekt befinden sich noch weiter hinten im Kapitel Ausblick auf Seite 55.

[1] *Perl* steht für *Practical Extraction and Report Language*, siehe [Perl 96].

3.2 Komponenten für die Textbearbeitung

Bei Betrachtung der Abbildung 3.1 auf der vorherigen Seite fällt sofort auf, daß das Kommando *rtftohtml* hier in gewisser Weise eine Sonderstellung einnimmt, obwohl es im Prinzip auch wie alle übrigen Module nur ein Teil der langen Pipeline-Kette ist. Da dieses Programm das Grundgerüst für die Umwandlung von RTF nach HTML darstellt, unterscheiden sich die Module, die in der Pipeline vor *rtftohtml* den Text bearbeiten von denen dahinter schon allein hinsichtlich der Art der verarbeiteten Daten:

Während die *Präprozessoren* Textmanipulationen direkt auf dem Ausgangsformat *Rich Text* durchführen und sich zum Beispiel mit Texteigenschaften wie Schriftfarbe und -größe sowie den Hypertext-Links beschäftigen, und *Postprozessoren*, die den von *rtftohtml* erzeugten HTML-Code noch nachbearbeiten. Zu dieser Gruppe gehören zum Beispiel die Skripte zur Aufbereitung von Tabellen und zur Erstellung der Navigationsleisten.

Allgemein kann man festhalten, daß alle Module, die solche Informationen bearbeiten, die beim Durchlauf durch *rtftohtml* einfach verschwinden würden (wie zum Beispiel die Farbinformation) oder die das Kommando nicht verstehen kann (das betrifft Erweiterungen wie Hypertext-Links und -Anker), als Präprozessoren in der Pipeline ablaufen müssen. Das bedeutet nun aber keineswegs, daß die Präprozessoren dafür noch keinen HTML-Code erzeugen können, viele tun das sogar. Man muß lediglich darauf achten, ihn so in den RTF-Text einzubetten, daß *rtftohtml* ihn erkennen kann und unverändert wieder in die Ausgabe übernimmt.

Interessanterweise ist diese Fähigkeit bereits vorgesehen worden (dazu muß der gewünschte Textbereich in der RTF-Datei nur als *versteckt* und *schattiert* gekennzeichnet werden), eine genaue Erläuterung zu diesem Punkt folgt etwas später auf Seite 41 bei der Erklärung der Funktionsweise des Programms.

Im folgenden werde ich nun in der Ablaufreihenfolge die gesamte Toolfolge kurz vorstellen und dabei auf einzelne wichtige Punkte genauer eingehen.

`title.pl`

> Dieses Modul hat – wie der Name schon vermuten läßt – nur die Aufgabe, den Dokumenttitel in den RTF-Text einzutragen. Weil *Edit* das dafür im Standard vorgesehene Schlüsselwort `\title` nicht verwendet, muß es hier nachträglich eingesetzt werden.
>
> Die einzige Schwierigkeit besteht darin, daß bei Verwendung der `-nav` Option von *rtfdtohtml* zunächst der Titel aus der angegebenen Beschreibungsdatei für den Dokumentbaum entnommen werden muß, da dort ja der Dokumenttitel explizit angegeben werden kann. Ansonsten wird einfach der Dateiname des Dokuments ohne die Kennung benutzt.

`lindent.pl`

Bei der Betrachtung der Einrücktiefe eines Textabsatzes wird normalerweise auch die Position des linken Rands mit berücksichtigt. Dabei ist jeweils eine Verschiebung des linken Rands nach rechts oder links um mehr als einen vorgegebenen Schwellenwert äquivalent zum Einfügen beziehungsweise Entfernen eines Tabulators am Zeilenanfang.

Da sich der später ablaufende Algorithmus[2] zur Bestimmung der aktuellen Einrücktiefe nur an der Anzahl der führenden Tabulatorzeichen orientiert, muß eine Änderung der linken Randeinstellung entsprechend durch Hinzufügen oder Entfernen von Tabulatoren am Zeilenanfang simuliert werden.

Wenn die Option `-no-li` bzw. `--no-lindent` angegeben worden ist, wird dieses Modul von *rtfdtohtml* aus nicht aufgerufen.

`preproc.sed`

Bei diesem Programm handelt es sich um ein einfaches *sed*-Skript, das eigentlich nur die Formatierung der RTF-Eingabedatei ein wenig abändert, so daß die nachfolgenden Skripte leichter damit zurechtkommen können. Insbesondere wird hier dafür gesorgt, daß in den Text eingefügte Bilder, Links oder Anker auf einer eigenen Zeile in der Eingabe stehen (was aber nicht gleichbedeutend mit einer Zeile im RTF-Dokument ist).

Zusätzlich wird noch die Angabe über die verwendete Zeichensatzkodierung in der Eingabedatei angepaßt. *Edit* verwendet dort immer das Schlüsselwort `\ansi`[3], obwohl das natürlich nicht richtig ist, da *OpenStep* eine eigene Zeichensatzkodierung verwendet. Neuere Versionen von *TextEdit* geben deswegen – eigentlich korrekterweise – an dieser Stelle `\next` als Schlüsselwort an. Das führt nun leider dazu, daß man keine andere Wahl hat, als beide in gleicher Weise zu behandeln und `\next` durch `\ansi` (oder umgekehrt) zu ersetzen.

`center.pl`

Eigentlich wäre es die Aufgabe von *rtftohtml*, sich um die Ausrichtung von Absätzen (linksbündig, rechtsbündig oder zentriert) zu kümmern, da das Kommando die entsprechenden RTF-Schlüsselworte kennt. Leider werden sie jedoch in der von mir verwendeten Version (außerhalb von Tabellen) offenbar vollständig ignoriert. Daher sucht dieses Skript in der Eingabe nach Ausrichtungswechseln und fügt dort die entsprechenden `<DIV align=...>` Elemente ein.

Falls der Benutzer explizit die Verwendung von HTML 2.0 verlangt hat (mit der `-html2` Option), wird hier statt dessen das Formatelement `<CENTER>` an

[2]Dieser ist Teil von *indent.pl*, das auch die Tabellenerstellung durchführt.
[3]Damit wird laut RTF-Spezifikation die Zeichensatzkodierung *ISO-Latin-1* ausgewählt.

Stelle von `<DIV>` eingesetzt, und rechtsbündige Absätze werden wie normaler, linksbündiger Text behandelt.

`color.pl`

Diese Modul befaßt sich mit der Umsetzung von Farben im RTF-Dokument in eine äquivalente HTML-Repräsentierung. Dabei wird aber nur die Schriftfarbe berücksichtigt, da die Hintergrundfarbe global eingestellt werden kann (siehe Seite 31).

Hierzu muß zunächst die Farbtabelle im RTF-Dokument analysiert werden, um zu jedem Eintrag eine entsprechende RGB-Farbbezeichnung in hexadezimaler Form (zum Beispiel `#ffff00` für Gelb) zu gewinnen. Anschließend werden ganz analog zum zuvor beschriebenen `center.pl` die Schriftfarbenwechsel herausgesucht und an diesen Stellen entsprechende `<FONT color=...>` Elemente eingesetzt.

`fsmap.pl`

Bei der Einbindung von *rtftohtml* stellte sich schnell als Schwachpunkt heraus, daß sich zwar über die Konfigurationsdateien bestimmten, fest vorgegebenen Schriftgrößen HTML-Formatelemente für die Ausgabe zuordnen lassen, dies jedoch nicht für ganze Größenbereiche (z.B. 10–14 Punkte) vorgesehen ist.

Da es wohl kaum zweckmäßig ist, sämtliche möglichen Schriftgrößen dort aufzuzählen, wurde dieses Modul entwickelt. Es bildet jeweils bestimmte Größenbereiche auf einen fest gewählten Repräsentanten für diese Größenklasse ab, dem dann in der Konfigurationsdatei von *rtftohtml* ein Formatelement (`<H1>`, `<H2>` usw.) zugeordnet werden kann.

`next.pl`

Die Aufgabe dieses Skripts ist, die wichtigen *NeXT*-spezifischen Erweiterungen des RTFD-Formats zu verarbeiten, also eingebettete Objekte, Hypertext-Links und -Markierungen (siehe Abschnitt 1.2 auf Seite 4). Da *rtftohtml* diese nicht erkennt und damit wie normalen Text behandelt, müssen sie schon vorher in die jeweils zugehörigen HTML-Darstellungen umgewandelt und dann im RTF-Text ersetzt werden.

Man darf dabei natürlich nicht vergessen, eventuelle Verweise auf EPS- und TIFF-Bilder sowie auf andere RTF- und RTFD-Dateien so anzupassen, daß sie am Ende auch auf die daraus resultierenden Umwandlungsergebnisse (also GIF/JPEG/PNG-Bilder bzw. HTML-Seiten) zeigen.

Zusätzlich werden auch noch die mit dem *TextLinker* erstellten Links und Marker erkannt. Diese werden unmittelbar in die „normale“ Darstellung (wie sie von *Edit* erzeugt wird) zurückübersetzt und genauso wie oben beschrieben weiterverarbeitet.

`patch.pl`

Im Laufe der Zeit stellte sich schrittweise heraus, daß auch *rtftohtml* noch nicht ganz fehlerfrei ist und eine ganze Reihe von Programmierfehlern enthält, die sich aber glücklicherweise alle durch einige leichte Modifikationen der RTF-Eingabe umgehen lassen. Die Aufgabe dieses Moduls ist es, diese Modifikationen durchzuführen.

Der größte Teil der Nachbesserungen, die in diesem Skript früher enthaltenen waren ist allerdings inzwischen wieder entfernt worden, da ich die entsprechenden Fehler jetzt direkt in *rtftohtml* korrigiert habe. Einige Ausführungen dazu befinden sich weiter hinten im Kapitel 4.

Der von den bisher beschriebenen Modulen vorverarbeitete RTF-Text wird nun an das Hilfsprogramm *rtftohtml* weitergeleitet, das daraus eine – zunächst vorläufige – HTML-Version erstellt. Um besser verstehen zu können, wie dieses Hilfsprogramm in das gewählte Modell integriert wurde, muß ich kurz darauf eingehen, wie *rtftohtml* arbeitet:

Zuerst wird die Eingabedatei in Schlüsselworte und normale Zeichen zerlegt, wobei jedem Zeichencode anhand einer Tabelle ein Zeichenname zugeordnet wird. Das ist notwendig, weil nicht alle Schriften in der Zeichensatzkodierung *ISO-Latin-1* vorliegen; dementsprechend gibt es zum Beispiel auch eine eigene Tabelle für die *Symbol* Schrift. Dies ist ein kleiner Ausschnitt aus der dazu gehörigen Datei *next-sym*[4]:

```
greater         >
question        ?
congruent       @
Alpha           A
Beta            B
```

Anschließend wird in einem zweiten Schritt die HTML-Ausgabe produziert. Dabei werden zunächst die zuvor gewonnenen Zeichennamen anhand einer Ausgabetabelle (*html-map*) in HTML-Text umgewandelt. Das Ergebnis ist in vielen Fällen wieder nur ein einzelnes Zeichen, kann aber bei Ersatzdarstellungen durchaus auch länger sein, wie die folgenden Zeilen demonstrieren:

```
greater         &gt;
question        ?
congruent       ~
Alpha           Alpha
Beta            Beta
```

[4] Die Dateinamen setzen sich immer aus der allgemeinen Zeichensatzkodierung (*next*, *ansi*) und der Bezeichnung für die Schriftart zusammen (hier *sym* für *Symbol*).

Selbstverständlich kann (und muß) diese Tabelle auch noch nachträglich angepaßt werden, wenn im HTML-Standard neue Ersatzdarstellungen (`α` für α usw.) eingeführt werden. Die Formatierung des auf diese Weise erhaltenen Texts wird anschließend von den Einstellungen in der *html-trans* Datei festgelegt.

Der eigentlich Zweck dieser Datei ist, die Abbildung von RTF-Absatzstilen auf die entsprechenden HTML-Formatelemente festzulegen, was aber aufgrund der schon erwähnten fehlenden Fähigkeiten von *Edit* in diesem Bereich leider überhaupt nicht ausgenutzt werden kann. Alternativ erlaubt *rtftohtml* die direkte Zuordnung von HTML-Formatelementen zu bestimmten Schriftarten und -größen oder Textauszeichnungen wie z.B. fett, kursiv, unterstrichen oder versteckt.

```
# Text Match Table
# Format:
# "Font",FontSize,Match,Mask,"TextStyleName"

# hidden/shadow -> literal
#    v^bDWUHACSOTIB,v^bDWUHACSOTIB
"",0,00000010010000,00000010010000,"_Literal"

# Special: Any text in ‘Quote_Font’ font -> literal
"Quote_Font",0,00000000000000,00000000000000,"_Literal"

# Discard special characters in NeXTdoc font
"NeXTdoc",0,00000000000000,00000000000000,"_Discard"

# monospace fonts -> tt
"Courier",0,00000000000000,00000000000000,"tt"
"Monoco",0,00000000000000,00000000000000,"tt"
"Ohlfs",0,00000000000000,00000000000000,"tt"

# some special font sizes
#     v^bDWUHACSOTIB,v^bDWUHACSOTIB
"",60,00000000000000,00000000000000,"h0"
"",48,00000000000000,00000000000000,"h1"
"",38,00000000000000,00000000000000,"h2"
"",32,00000000000000,00000000000000,"h3"
"",28,00000000000000,00000000000000,"h4"
```

Abbildung 3.2: Auszug aus der Datei *html-trans*

Die Abbildung 3.2 zeigt einen kleinen Ausschnitt aus dieser Datei, in dem man manche Aspekte wiederfinden kann, auf die ich schon bei den Erläuterungen zu einigen Modulen hingewiesen habe.

Viele der Präprozessoren verwenden zum Beispiel die Schriftattribute versteckt und schattiert (*hidden/shadow*), um direkt HTML-Text an der betreffenden Stelle in der Ausgabe von *rtftohtml* plazieren zu können. Auf diese Weise können damit praktisch HTML-Formatanweisungen mitten unter den *Rich Text* gemischt werden.

Auf demselben Mechanismus beruht übrigens auch das Verhalten der über die Umgebungsvariable `RTFDQUOTEFONT` einstellbaren Schriftart, mit der HTML-Code auch schon gleich in *Edit* eingefügt werden kann. Ebenso kann man ganz einfach vorgegebene Schriftarten sogar ganz aus der Ausgabe verschwinden lassen, was hier für die Schrift *NeXTdoc* verwendet wird.

Auch die Liste der als nicht-proportional definierten Schriftarten wird hier festgelegt und kann auch bei Bedarf vom Benutzer entsprechend erweitert werden. Diese Schriften erzeugen dann bei der Verwendung vorformatierte Textbereiche.

Da die Einträge in dieser Tabelle (unter anderem) nach Schriftart und -größe ausgewählt werden und man als Größe nur eine feste Zahl (oder 0 für beliebige Größen) angeben kann, wird auch deutlich, warum die mühsame Abbildung auf einzelne Schriftgrößen durch `fsmap.pl` notwendig ist. Auch die Liste der festgelegten Repräsentanten für jeden Größenbereich findet sich hier wieder.[5]

Anschließend wird die Ausgabe des Kommandos *rtftohtml* in der zuvor beschriebenen Pipeline-Kette durch eine Reihe von Postprozessoren weitergeschickt, um verschiedene Aspekte noch einmal nachbearbeiten zu können. Zur Zeit existieren vier solche Postprozessoren:

`fixed.pl`

> Dieses Modul beschäftigt sich mit der Zusammenfassung von mehrzeiligen Textbereichen zu einem großen, vorformatierten Abschnitt. Voraussetzung dafür ist, daß keine der betrachteten Zeilen ein Zeichen (außer Zwischenraum) in einer proportionalen Schrift enthält.
>
> Außerdem muß man hier darauf achten, daß nicht versehentlich ein Bild in einen solchen Bereich mit aufgenommen wird, da das nach dem HTML-Standard nicht zulässig ist. Dies könnte durchaus passieren, da einem Bild ist ja keine Schriftart zugeordnet ist und somit für das Skript auch kein Schriftartwechsel zu erkennen ist.
>
> Anschließend können wiederum mehrere derartige, aufeinanderfolgende Abschnitte zu einem einzigen zusammengefaßt werden, sofern sie nur durch Zwischenraum (wie zum Beispiel Leerzeilen) voneinander getrennt sind.

`indent.pl`

> Hierbei handelt es sich um das mit Abstand umfangreichste und komplexeste Skript zur Nachbearbeitung. Es kümmert sich im wesentlichen um die

[5] *h0* ist keine gültige Überschrift in HTML, es wird von einem der Postprozessoren später durch *h1* ersetzt.

(schon im Abschnitt 2.2.4 auf Seite 14 erläuterte) Umwandlung von einzelnen, durch Tabulatorzeichen getrennte Spalten in Tabellen oder Listen und kann dabei auch mehrere aufeinanderfolgende Tabellen und/oder Listen zusammenfassen.

Hier muß man allerdings berücksichtigen, daß die vom zuvor betrachteten Modul `fixed.pl` erstellten vorformatierten Bereiche natürlich auch Tabulatorzeichen enthalten können (z.B. bei Quelltext), diese aber nicht beachtet werden dürfen.

Alternativ dazu können auch einzelne Absätze einfach nur ein- oder ausgerückt werden, falls es sich in der Eingabe nur um normalen, eingerückten Text handelt.

`heading.pl`

Wie der Name dieses Moduls schon richtig vermuten läßt, werden hier noch einmal die Überschriften nachbearbeitet und Ansprungpunkte für den Index hinzugefügt (falls die `-index` Option angegeben wurde). Da automatisch alle Überschriften im Index auftauchen, muß in diesem Fall jede einzelne eine eindeutige Markierung bekommen, über die sie dann später aus der Index-Datei aufgerufen werden kann.

Außerdem werden die alten Einträge zu dem gerade bearbeiteten Dokument aus dem Index-Cache *.index.in* entfernt und an deren Stelle die neuen Verweise in die Datei eingetragen.

Auch die Abbildung zwischen den `<H1>`...`<H6>` und `<FONT size=...>` Formatelementen, deren Verwendung über die Option `-head` kontrolliert werden kann, wird hier durchgeführt (siehe dazu die Erläuterungen im Abschnitt 2.3.1 auf Seite 23).

`nav.pl`

Dieses letzte Modul schließlich ist für die automatische Generierung von Navigationsleisten am Anfang und Ende jeder erzeugten HTML-Datei verantwortlich, sofern der Benutzer die `-nav` Option und als Argument eine entsprechende Textdatei mit der Beschreibung der Struktur der gesamten Dokumenthierarchie angegeben hat.

Das wesentliche Problem, das in diesem Skript zu lösen ist, liegt darin, diese Datei zu analysieren und die entsprechenden Verweise auf das nächste, vorhergehende und logisch übergeordnete Dokument richtig zuzuordnen. Sollte es eine Index-Datei geben, wird sie hier ebenfalls mit in die Navigationsleiste aufgenommen.

3.3 Komponenten für die Grafikumwandlung

Im Bereich der Grafikumwandlung habe ich im Prinzip denselben Ansatz wie bei der Textbearbeitung verfolgt, hier übernimmt aber – wie schon erwähnt – nicht *rtfdtohtml* selbst die Ablaufsteuerung, sondern das Skript *anytoimg.sh*, daß bei Bedarf von *rtfdtohtml* aus für die Grafikumwandlung gestartet wird. Durch die auf diese Weise definierte Arbeitsteilung bleiben beide Programme noch relativ klein und übersichtlich.

Von diesem Skript aus werden dann im Verlauf des Umwandlungsvorgangs für jedes Bild die jeweiligen speziellen Konvertierungsprogramme aufgerufen, wobei immer nach dem aktuellen Eingabeformat und dem gewünschten Ausgabeformat differenziert wird. Dazu gibt es eine ganze Reihe von kleinen Shell-Skripten, die alle Namen der Form besitzen:

Eingabeformat`to`*Ausgabeformat*`.sh`

Zur Zeit werden diese beim Installationsvorgang als symbolische Links eingerichtet, die dann auf einen der wirklichen Grafikfilter verweisen. Dazu muß bei der Installation ausgewählt werden, welche Hilfsprogramme jeweils für die Umwandlung verwendet werden sollen:

```
# uncomment exactly one of the following lines:
# EPS = e2t#    use eps2tiff (NeXTSTEP only)
# EPS = gs#     use ghostscript & ppm utils
# EPS = img#    use ghostscript & ImageMagick

# uncomment exactly one of the following lines:
# TIFF = oif#   use ppm utils & OmniImageFilter (NeXTSTEP only)
# TIFF = ppm#   use ppm utils
# TIFF = img#   use ImageMagick & tiff utils & tiff2png

ln -s "epstogif_$(EPS).sh" "$(libbindir)/epstogif.sh"
ln -s "epstogif_$(EPS).sh" "$(libbindir)/epstopng.sh"
ln -s "epstogif_$(EPS).sh" "$(libbindir)/epstojpeg.sh"
ln -s "tifftogif_$(TIFF).sh" "$(libbindir)/tifftogif.sh"
ln -s "tifftogif_$(TIFF).sh" "$(libbindir)/tifftopng.sh"
ln -s "tifftogif_$(TIFF).sh" "$(libbindir)/tifftojpeg.sh"
```

Wie man hier gut erkennen kann, wäre die Unterscheidung nach dem Ausgabeformat bei der Skriptauswahl eigentlich nicht mehr notwendig. Das liegt daran, daß in der aktuellen Version die Umwandlungsskripte im Prinzip beliebige Ausgabeformate erzeugen können sollten[6]. Dennoch ist die ursprüngliche Aufteilung

[6] Allerdings sind nicht für alle Skripte auch alle Ausgabeformate implementiert.

immer noch beibehalten worden, so daß theoretisch für jede dieser Kombinationen ein eigener Umwandlungsfilter existieren könnte.

Für jedes der beiden Eingabeformate TIFF und EPS stehen nun jeweils drei unterschiedliche Grafikumwandlungsskripte bereit, die mit verschiedenen Hilfsprogrammen arbeiten. Auf diese Weise ist sichergestellt, daß auch Systeme verwendet werden können, auf denen bestimmte Hilfsprogramme nicht vorhanden sind. Daß die wirklich verwendeten Grafikfilter alle *togif* im Namen enthalten, liegt nur daran, daß GIF das erste implementierte Format gewesen ist, und hat keine weitere Bedeutung mehr.

Das Programm *anytoimg.sh* geht nun der Reihe nach alle Bilddateien durch, die sich in dem zum aktuellen Dokument gehörigen Katalog befinden. Dieser existiert immer, da das Kommando ja nur für RTFD-Dokumente aufgerufen wird und diese stets aus einem Katalog bestehen, der mindestens den Dokumenttext (in der Datei *TXT.rtf*) und alle darin eingebetteten Objekte als Dateien enthält:

```
$ ls -l Example.rtfd
TXT.rtf         build.tiff          paste.eps
amiga.gif       button_smile.jpeg
```

Für jede einzelne Bilddatei muß *anytoimg.sh* dabei, immer basierend auf der Dateikennung, entscheiden, ob eine Umwandlung erforderlich ist oder nicht. Falls ja, startet es das entsprechende Umwandlungsskript für das betreffende Format.

Die sechs im Moment zur Verfügung stehenden Umwandlungsfilter sind in der nun folgenden Auflistung jeweils kurz beschrieben, wobei insbesondere auf spezielle Probleme und Eigenarten eingegangen wird. Alle Skripte sind echte Filter, das heißt, sie können die Bilddaten von der Standardeingabe lesen und geben das Umwandlungsresultat auf der Standardausgabe aus. Für viele der hier erwähnten Hilfsprogramme befindet sich auch eine nähere Beschreibung im Abschnitt 1.4 auf Seite 6.

`epstogif_e2t.sh`

Dieses Shell-Skript benutzt das Hilfsprogramm *eps2tiff*, das unter Verwendung des *OpenStep* Window-Servers die Eingabedatei in das TIFF-Format konvertieren kann. Dieses Programm wird benutzt, um in einem ersten Schritt aus der Eingabe eine TIFF-Datei zu erzeugen.

Voraussetzung hierfür ist jedoch, daß dieses Kommando auch eine Verbindung zum lokalen Window-Server herstellen kann, was zur Folge hat, daß es nur direkt am Bildschirm eines *NeXTStep*/*OpenStep*-Systems funktioniert. Ein Prozeß, der nach einer Anmeldung über *telnet* oder *rlogin* gestartet wurde, darf normalerweise nicht auf den Window-Server zugreifen.

Im zweiten Schritt wird dann noch das für die Umwandlung von TIFF in das verlangte Ausgabeformat vorgesehene Skript `tiffto`*Ausgabeformat*`.sh` aufgerufen, um die gewünschte Ausgabe zu erzeugen. Ein Teil der Aufgabe wird also einfach an ein bereits vorhandenes Programm weitergeleitet.

epstogif_gs.sh

Da *rtfdtohtml* ja eigentlich Plattform-unabhängig funktionieren soll, darf man sich natürlich nicht darauf verlassen, daß dem Benutzer immer ein *OpenStep* Window-Server zur Verfügung steht. Daher kann als Alternative dazu das Programm *Ghostscript* verwendet werden, eine portable und frei verfügbare Implementierung eines PostScript-Interpreters. Allerdings hängt hierbei die Qualität der Ausgabe zu einem wesentlichen Teil von den auf dem System zur Verfügung stehenden PostScript-Schriften ab, so daß unter Umständen nicht alle Beschriftungen ganz korrekt wiedergegeben werden können.

Dieses Programm enthält nun unter anderem ein Modul *pstoppm.ps*, das EPS-Grafiken in das PPM-Format (*Portable Pixmap File Format*) konvertieren kann. Allerdings erhält man dabei als Ausgabe immer eine ganze Seite – typischerweise in Größe DIN-A4 – auf der dann das gewünschte Bild irgendwo positioniert ist. Glücklicherweise kann man diese Positionsangabe aber ohne großen Aufwand direkt der EPS-Datei entnehmen, die immer eine Zeile der Form:

```
%%BoundingBox: 161 492 271 571
```

mit den entsprechenden Koordinaten auf der Seite enthalten muß.

Mit diesen Informationen kann das das Programm *pnmcut* aus dem *Netpbm* Paket (siehe dazu die Beschreibung auf Seite 7) diesen Bereich der Seite exakt ausschneiden. Anschließend muß nur noch dieser Ausschnitt in einem zweiten Schritt in das Zielformat umgewandelt werden, wozu dann wiederum eines der *Netpbm*-Kommandos verwendet wird.

epstogif_img.sh

Dieses Skript ist eigentlich die einfachste der drei Varianten zur Umwandlung von EPS-Grafiken und besteht im wesentlichen nur aus einem Aufruf des *convert* Programms von *ImageMagick*. Da dieses allerdings intern wiederum Ghostscript für die Umwandlung der PostScript-Datei verwendet, setzt auch diese Version voraus, daß Ghostscript auf dem System installiert ist.

Leider stellte sich später beim Testen heraus, das manche Grafikprogramme offenbar EPS-Dateien erzeugen, die mehrere *BoundingBox* Angaben enthalten können (diese beschreiben die Position und Größe des Bilds auf der Seite). In diesem Fall wählt *ImageMagick* fälschlicherweise immer die letzte dieser Angaben aus, was dann in den meisten Fällen anstelle des gewünschten Bilds nur eine weiße Fläche ergibt.

Um dieses Problem zu vermeiden, muß man die Eingabedatei zuerst noch auf mehrfach auftretende *BoundingBox* Zeilen untersuchen und alle außer der ersten aus der EPS-Datei entfernen, bevor sie an *convert* zur Umwandlung übergeben wird.

`tifftogif_img.sh`

Das Problem, ein TIFF-Bild in ein anderes Bitmapgrafikformat zu konvertieren, scheint auf den ersten Blick einfacher zu sein als die Umwandlung eines Vektorgrafikformats wie EPS. Bei näherer Betrachtung zeigen sich aber leider einige Details, die dieses Problem deutlich komplizierter werden lassen, als es zunächst den Anschein hat:

So kann eine Grafik im TIFF-Format zum Beispiel getrennt von der Farbinformation auch noch eine Transparenzinformation enthalten (*Associated Alpha*), und nur wenige Programme können damit wirklich korrekt umgehen.

Das Problem wird auch dadurch nicht gerade einfacher, daß eine einzelne TIFF-Datei mehrere Versionen eines Bilds enthalten kann, zum Beispiel in verschiedenen Farbtiefen oder schwarz-weiß und farbig. In diesem Fall muß zuerst das sogenannte *TIFF Directory* nach der geeigneten Version[7] des Bilds abgesucht und diese dann als Einzelbild aus der Datei extrahiert werden.

Anschließend wird danach unterschieden, ob das Bild Transparenz enthält oder nicht. Falls nein, kann es direkt mit *convert* in das gewünschte Zielformat umgewandelt werden. Andernfalls wird die Sache etwas komplizierter, da auch *ImageMagick* immer noch Probleme mit manchen schwarz-weißen, transparenten TIFF-Bildern hat. Um das zu umgehen, wird das Bild zuerst mit *tiff2png* in das PNG-Format gebracht.

Schließlich ergibt sich noch die Schwierigkeit, daß die typischen Formate GIF und JPEG die Transparenzinformation nicht aufnehmen können, also muß zuerst mit *montage* (ein weiteres *ImageMagick* Programm) der transparente Anteil entfernt werden, bevor es dann endlich in das Zielformat umgewandelt werden kann.

`tifftogif_oif.sh`

Dieses Skript versucht, die Fähigkeiten des *OpenStep*-Programms *OmniImageFilter* auszunutzen, das sich beim Pasteboard-Server als Image-Filter registriert und dann praktisch im Auftrag des Pasteboard-Servers Grafikumwandlungen durchführen kann. Dazu müssen die Bilddaten zunächst auf einem privaten Pasteboard abgelegt werden, das anschließend die Daten im neuen Format liefern kann, wenn ein entsprechender Image-Filter vorhanden ist. Natürlich funktioniert diese Lösung ganz analog zu `epstogif_e2t.sh` (siehe Seite 46) nur bei direkter Anmeldung an einem *NeXTStep*- bzw. *OpenStep*-System.

[7] Dabei wird im Moment immer die Version des Bilds mit der höchsten Farbanzahl verwendet.

Da man von einem Shell-Skript aus nicht direkt mit dem Pasteboard-Server kommunizieren kann, setze ich dafür zwei kleine Hilfsprogramme aus den Vorlesungsunterlagen zu Informatik C ein, *put* und *trafo*, die Daten auf das Pasteboard bringen und von dort wieder abholen können.

Weil der *OmniImageFilter* nicht direkt das gewünschte Ausgabeformat produzieren kann, muß man wieder einmal einen Zwischenschritt über das PPM-Format wählen und dieses anschließend mit *ppmtogif* in das GIF-Format umwandeln (andere Formate sind hier im Moment nicht implementiert). Der Programmtext dazu sieht verblüffend einfach aus:

```
if [ $# = 0 ]; then
    put -tiff "$pbname"
else
    put -tiff "$pbname" <"$1"
fi

trafo "$pbname" "$pbtype" | ppmtogif -interlace -sort
```

Das ganze funktioniert aber nur, falls das auf diesem Weg erzeugte PPM-Bild nicht zu viele oder zu wenige Farben hat, weil sonst *ppmtogif* eine Fehlermeldung ausgibt bzw. sofort abstürzt (und einen core-dump hinterläßt). Wenn ein solcher Fall eintritt, wird automatisch versucht, die Anzahl der Farben zu reduzieren, um das Problem zu vermeiden.

Da das Zwischenformat PPM keine Transparenz enthalten kann, erscheinen alle transparenten Pixel nach der Umwandlung in Schwarz. Außerdem wird bei TIFF-Dateien mit mehreren Bildern stets das erste verwendet, auch wenn das nur schwarz-weiß ist.

`tifftogif_ppm.sh`

Dieses Skript benutzt ausschließlich Kommandos aus dem *Netpbm* Paket, um eine Umwandlung vom TIFF- in das GIF-Format durchzuführen (auch hier steht im Moment kein anderes Format zu Verfügung). Es ist das kürzeste der für die Grafikumwandlung implementierten Module und besteht (abgesehen von Kommentar) nur aus der Zeile:

```
tifftopnm ${1+"$1"} | ppmtogif -interlace -sort
```

Da auch hier ein Zwischenschritt unausweichlich ist, gelten die oben beschriebenen Probleme im Bezug auf Transparenz und TIFF-Dateien mit mehreren Bildern ebenso für dieses Skript.

3.4 Der TextLinker

Diese Applikation hat mit der ursprünglichen Aufgabenstellung, HTML-Code aus *Rich Text* zu erzeugen, nur wenig zu tun. Die Aufgabe des *TextLinkers* ist es, das *TextEdit* Programm um fehlende Funktionalität zu erweitern: Er soll dem Benutzer wie in *Edit* die Möglichkeit bieten, Hypertext-Links auf einfache Weise in den Text einzubinden. Dabei habe ich versucht, die schon vertraute Benutzerschnittstelle, die *Edit* hierfür verwendet, soweit wie möglich nachzubilden.

Die Verwendung dieser Applikation habe ich bereits im Abschnitt 2.2.7 auf Seite 19 vorgestellt, hier werde ich nur auf einige Aspekte der Implementierung eingehen. Um die gestellte Aufgabe erfüllen zu können, muß der *TextLinker* allerdings ziemlich tief in das *OpenStep*-System integriert werden, damit eine sinnvolle Interaktion mit *TextEdit* überhaupt möglich wird. Weil er sich für den Benutzer im Prinzip genauso wie ein Dialogfenster innerhalb des Editors verhalten soll und dementsprechend eine dazu passende grafische Oberfläche besitzen muß, ergibt sich zwangsläufig, daß die Implementierung in Objective C erfolgen muß.

Damit unterscheidet sich der *TextLinker* deutlich von den übrigen für diese Diplomarbeit entwickelten Programmen, die allesamt nur auf verschiedenen Skriptsprachen (Perl, sed und der Shell) aufbauen. Das ist aber nichts weiter als eine logische Konsequenz der sehr unterschiedlichen Anforderungen: Für den *TextLinker* ist eine Anbindung an die grafische Oberfläche (und speziell das *Pasteboard*) von zentraler Bedeutung, während dieser Punkt bei *rtfdtohtml* überhaupt keine Rolle spielt.

Um bei den nachfolgenden Erläuterungen nicht zu weit ausholen zu müssen, setze ich dabei voraus, daß der Leser schon mit den Kommunikationsmechanismen der *OpenStep*-Benutzeroberfläche und den dafür notwendigen Programmiertechniken vertraut ist. Aufgrund des sehr speziellen Aufgabenfelds kommen hier auch einige neue Begriffe vor, deren Bedeutung ich als bekannt voraussetzen muß. Die Abbildung 2.1 auf Seite 19 zeigt die Einbindung des *TextLinkers* in das System.[8]

Die grundlegende Idee ist eigentlich relativ einfach:

Da man die Informationen über einen Link nicht mehr in der RTF-Datei selbst unterbringen kann, muß man einen anderen Weg finden, sie im Text abzulegen. *TextEdit* bietet dem Benutzer ja bereits die Möglichkeit, beliebige Dateien in ein RTFD-Dokument einzubetten.[9]

Daher liegt es nahe, einfach einen neuen Dateityp zu definieren, der alle für einen Hypertext-Link (bzw. einen Hypertext-Anker) benötigten Informationen in einer

[8] PBS bezeichnet in dieser Abbildung den *Pasteboard Server*.

[9] Dies ist keine spezielle Eigenschaft von *TextEdit*, sondern eine generelle Fähigkeit des darin verwendeten NSText- bzw. NSTextView-Objekts.

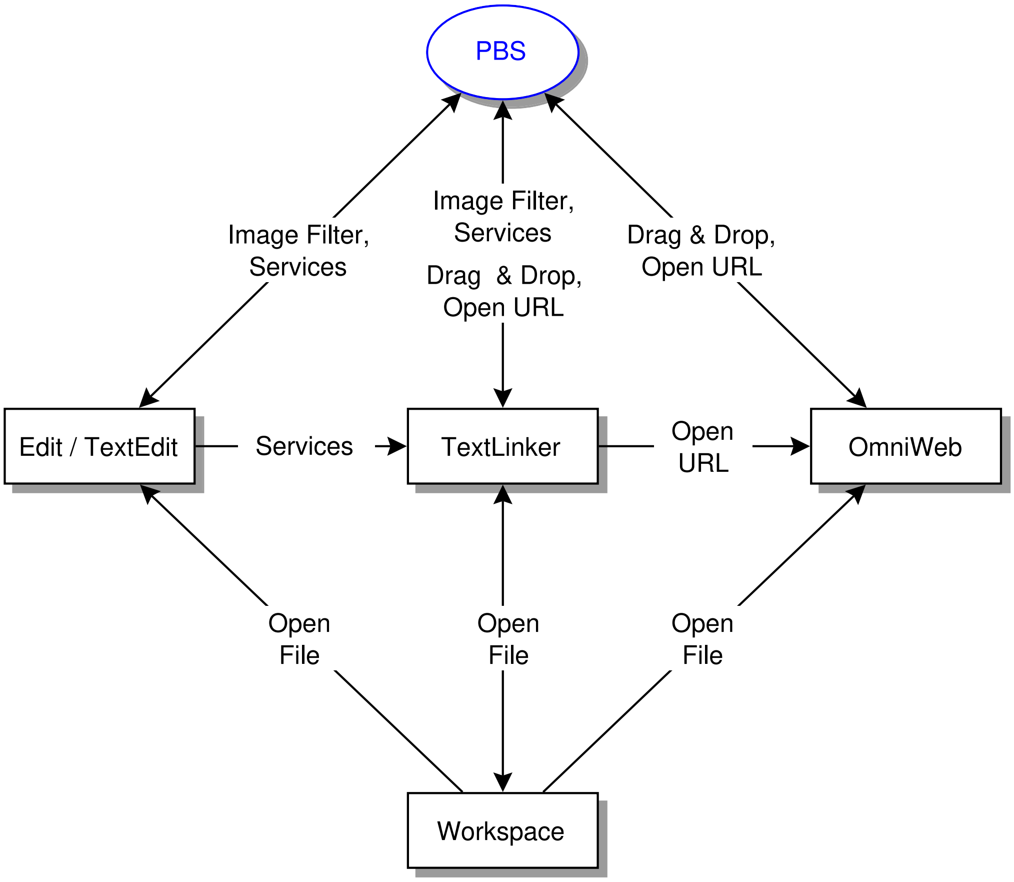

Abbildung 3.3: Einbindung des TextLinkers in das *OpenStep*-System

Form enthält, die bei der Umwandlung mit wenig Aufwand wieder zerlegt werden kann. Ich habe hierfür ein ganz einfaches Textformat gewählt:

`link:` *Ziel des Verweises (Datei oder URL)*
`mark:` *Anker im Zieldokument (optional)*
`name:` *Name dieses Ankers (optional bei einem Link)*

Dabei kann ein Link sogar auch gleichzeitig ein Anker sein. Bei einem reinen Anker entfallen die ersten beiden Zeilen, so daß nur der `name:` Eintrag übrig bleibt. Da bei *OpenStep* ein Dateityp nur über die Kennung identifiziert wird, müssen diese Dateien immer die beiden Kennungen *.tlink* (für einen Link) bzw. *.tmark* (für einen Anker) besitzen.

Dies ist theoretisch bereits völlig ausreichend, um die Verwendung von Hypertext-Links in *TextEdit* zu ermöglichen und benötigt kein weiteres Hilfsprogramm; die Erkennung dieser Links muß jedoch in das Modul `next.pl` von *rtfdtohtml* eingebaut werden (siehe Seite 40). Das Ziel ist natürlich, dem Benutzer die Erstellung und

Verwaltung dieser Dateien abzunehmen und möglichst einfach zu gestalten. Dafür ist nun der TextLinker entwickelt worden.

Die Implementierung gliedert sich dabei im wesentlichen in drei Schritte:

- Editiermöglichkeit über ein Dialogfenster

  Damit das Einfügen und Editieren solcher Hypertext-Link-Dateien möglichst einfach wird, bietet es sich an, diese Funktionalität direkt innerhalb von *TextEdit* im Services-Menü anzubieten. Über dieses Menü können anderen Applikation Dienste angeboten werden, die sich typischerweise auf die aktuelle Selektion beziehen und diese in bestimmter Weise bearbeiteten.

  Der *TextLinker* bietet nun einen Service an, der in den selektieren Text automatisch eine solche Datei einfügen kann. Die dazu notwendigen Informationen werden dabei über ein eigenes Dialogfenster (siehe Abbildung 2.1 auf Seite 19) abgefragt. Um diesen Service auch wirklich von anderen Programmen aus nutzen zu können, muß er in die Datei *CustomInfo.plist* eingetragen werden, etwa so:

  ```
  { NSMessage = editLink;
    NSPortName = TextLinker;
    NSSendTypes = ( "NeXT RTFD pasteboard type",
                    NSRTFPboardType );
    NSReturnTypes = ( "NeXT RTFD pasteboard type" );
    NSMenuItem = { default =
                   "TextLinker/Edit or Insert Link"; };
    NSTimeout = 600000;
  }
  ```

  Diese Beschreibung legt fest, welche Applikation den Service erbringt, welche Ein- und Ausgabetypen er unterstützt (hier RTF und RTFD) und welchen Menüeintrag er im Services-Menü bekommt. Damit das auch funktioniert, muß der *TextLinker* die Methode *editLink* implementierten, die das Dialogfenster anzeigt und vom Benutzer ausfüllen läßt. Anschließend werden die Information daraus in einer *.tlink*-Datei abgelegt und diese in den Rich Text eingefügt.

  Ganz analog kann man genau dasselbe auch für einen Hypertext-Anker implementierten und erhält so eine einfach zu bedienende Schnittstelle zum Editieren der Verknüpfungsdateien.

- Grafische Darstellung der Links und Anker im Text

  Das Aussehen von in den Text eingefügten Objekten hängt ganz allgemein von zwei Faktoren ab. Falls es sich direkt um ein bekanntes Grafikformat handelt (also EPS oder TIFF), oder sich der Dateiinhalt mit Hilfe des Pasteboard-Servers in eines dieser beiden Formate umwandeln läßt, wird das daraus erzeugte Bild im Text an dieser Stelle dargestellt. Wenn eine Umwandlung in ein Bild auf diesem Weg nicht möglich ist, wird statt dessen nur das Icon der Datei (wie es auch im *Workspace* erscheint) angezeigt.

Um das Aussehen der Hypertext-Links und -Anker zu kontrollieren, kann man also entweder den TextLinker als Applikation für die beiden Dateitypen registrieren und für sie die entsprechenden Icons hinterlegen, oder man versucht, für sie einen Grafikfilter beim Pasteboard-Server einzutragen. Da jedoch ein Icon stets eine feste Größe von 48 mal 48 Pixeln besitzt, wäre die erste Variante im Text viel zu auffällig.

Also bleibt nur noch die Möglichkeit übrig, dafür zu sorgen, daß *TextEdit* den Dateiinhalt als Bild interpretieren und dieses dann anzeigen kann. Dafür muß ein eigener sogenannter *Filter-Service* beim *Pasteboard* registriert werden, der als Eingabe Dateien mit den Kennungen *.tlink* und *.tmark* akzeptiert. Dazu muß nur die *CustomInfo.plist* Datei um die folgenden Zeilen ergänzt werden:

```
{ NSFilter = filter;
  NSPortName = TextLinker;
  NSSendTypes = ( NSTypedFilenamesPboardType:tlink,
                  NSTypedFilenamesPboardType:tmark );
  NSReturnTypes = ( NSTIFFPboardType );
}
```

Die Implementierung für diesen Teil ist noch relativ einfach zu realisieren, da nicht wirklich der Dateiinhalt in ein Bild umgewandelt werden muß (was ja wohl auch keinen Sinn macht). Statt dessen liefert das Programm einfach immer dieselben TIFF-Grafiken für jede Eingabedatei zurück: Ein kleines blaues Rautensymbol für einen Hypertext-Link (Dateikennung *.tlink*) und ein gelbes für einen Anker (*.tmark*).

- Verhalten bei Doppelklick auf das Link-Symbol

  Wenn der Benutzer ein in den Text eingebundenes Objekt mit der Maus doppelt anklickt, ist das für *TextEdit* eine Aufforderung, die betreffende Datei über den *Workspace* öffnen zu lassen. Dabei sucht der *Workspace* nach einer für diesen Dateityp registrierten Applikation, startet sie falls nötig, und läßt die Dateien dann von diesem Programm öffnen.

  Wenn man nun mit Doppelklick einen Link verfolgen können möchte, ist es notwendig, den *TextLinker* wenigstens für die auf Seite 51 beschriebene Dateikennung *.tlink* zu registrierten, damit man eine Nachricht vom *Workspace* erhält, sobald der Benutzer einen solchen Link im Editor doppelt angeklickt hat. Einen Anker anzuklicken ist natürlich ebenfalls möglich, dann wird aber die Beschreibungsdatei ganz normal in einem neuen Editorfenster geöffnet, weil keine dazu passende Applikation existiert.

Nun gibt es zwei verschiedene Möglichkeiten:

Falls sich der Verweis auf eine lokale Datei bezieht, wird der Dateiname einfach wieder zurück an den *Workspace* geschickt mit der Anweisung, ihn in der zugehörigen Applikation zu öffnen. Bei einem Verweis auf eine RTF-Datei könnte das zum Beispiel wieder der Editor selbst sein.

Bei einem Verweis auf eine beliebige URL läuft der Vorgang im Prinzip ähnlich ab, wird aber nicht über den *Workspace* sondern wieder über das *Pasteboard* abgewickelt. Voraussetzung dafür ist, daß irgendein Programm auf dem System einen Service anbietet, der das Öffnen einer URL ermöglicht. Hierfür wird die URL zunächst zum *Pasteboard* übertragen und dann der entsprechende Service über `NSPerformService()` automatisch ausgelöst.

Die Voreinstellung für diesen Fall, die vom *TextLinker* normalerweise verwendet wird, ist der Service *Open URL* von *OmniWeb*.

Kapitel 4

Ausblick

Bei einer abschließenden Betrachtung kann man feststellen, daß von den in der Einleitung auf Seite 5 dargestellten Entwicklungszielen alle wesentlichen Punkte erreicht werden konnten:

rtfdtohtml bietet eine einfach zu bedienende und dabei Plattform-unabhängig Möglichkeit zur Erstellung von HTML-Dokumenten aus RTFD-Quelldateien, wobei auch eine automatisierte Bearbeitung von vielen Dateien keine Schwierigkeiten bereitet. In diesem Bereich stellte sich auch die Fähigkeit zur automatischen Erstellung von Navigationsleisten als eine sinnvolle Ergänzung heraus. Die Qualität der generierten HTML-Seiten ist dabei so gut, daß eine spezielle Nachbearbeitung von Hand normalerweise nicht mehr erforderlich ist, wenn auch aufgrund des eingeschränkten Funktionsumfangs von *Edit* längst nicht alle Möglichkeiten, die das HTML-Format bietet, auch ausgenutzt werden können. Durch den sehr modularen Aufbau des Programms lassen sich jedoch leicht eigene Änderungen und Erweiterungen vornehmen, so daß es in jedem Fall an zukünftige Entwicklungen angepaßt werden kann.

Im Vergleich zu der kommerziellen Alternative *LatinByrd*, einer reinen *OpenStep*-Applikation, bietet *rtfdtohtml* einen etwa vergleichbaren Funktionsumfang, allerdings stellenweise mit einem anderen Schwerpunkt. *LatinByrd* setzt dabei konsequent auf eine ausgereifte grafische Benutzeroberfläche, wodurch das Programm aber nur noch unter *OpenStep* lauffähig und eine Portierung praktisch ausgeschlossen ist. Die wesentlichen Probleme bestehen meiner Ansicht nach jedoch darin, daß zum einen durch die grafische Oberfläche wohl eine nicht-interaktive Bedienung kaum in Frage kommt, und zum anderen die Möglichkeiten für eigene Anpassungen und einfache Erweiterbarkeit schon deswegen sehr eingeschränkt sind, weil der Quelltext des Programms nicht zur Verfügung steht. Hinzu kommt, daß auch in der aktuellen Version von *LatinByrd* der erstellte HTML-Code nicht immer vollständig korrekt ist, auch wenn sich das durch die sehr tolerante Handhabung durch die Web-Browser nicht nachteilig bemerkbar macht.

Auf der anderen Seite bleibt allerdings (wie so oft) am Ende immer noch ein breiter Raum für weitere Verbesserungen:

Eine wesentliche Verbesserung hinsichtlich der erzeugten HTML-Ausgabe ließe sich noch durch den Einsatz der neuen *Cascading Style Sheets* (kurz CSS) erreichen, damit könnte man Absatzstile aus den RTF-Dokumenten direkt in CSS-Stildefinitionen übertragen. Leider ist das aber aufgrund der fehlenden Unterstützung durch die als Grundlage verwendeten Programme *Edit* und *TextEdit* nicht zu realisieren, da diesen die hierfür notwendigen Fähigkeiten zur Definition von benannten Absatzstilen fehlen.

Ein anderes bisher noch ungelöstes Problem ist, daß die *libtiff*, auf der die Unterstützung für das TIFF-Format in *ImageMagick* beruht, mit einigen auf *OpenStep*-Systemen erstellten TIFF-Bildern immer noch nicht umgehen kann. Dies liegt aber vermutlich daran, das sich manche Programme nicht vollständig an das in der TIFF-Spezifikation [TIFF 92] festgelegte Format halten, was dann später unweigerlich zu Fehlermeldungen bei der Umwandlung führt.

Ganz generell zeigte sich bei der Implementierung der Grafikumwandlung, daß jedes der eingesetzten Hilfsprogramme an der einen oder anderen Stelle Schwachpunkte oder Programmierfehler hatte. Diese wirkten sich zwar am Ende nicht nachteilig auf die Funktionsfähigkeit von *rtfdtohtml* aus, da sich die dadurch auftretenden Probleme in allen Fällen relativ einfach umgehen ließen, aber ich war doch manchmal sehr erstaunt darüber, wie solche Fehler über lange Zeiträume unentdeckt bleiben konnten. Andererseits ist es natürlich ein bekanntes Phänomen, daß ein hinreichend komplexes Programm niemals vollständig fehlerfrei sein wird.

Wer mit den von mir verwendeten Hilfsprogrammen vertraut ist, dem ist vielleicht schon aufgefallen, daß ich bei der Implementierung mit Absicht ausschließlich auf nicht-kommerzielle Hilfsprogramme gesetzt habe, die auch als Quellcode verfügbar sind. Diese lassen sich meist ohne Probleme auf jedem beliebigen System übersetzen, was auch zu der angestrebten Plattform-unabhängig beiträgt, und sie lassen sich bei Bedarf sogar noch an die eigenen, speziellen Bedürfnisse anpassen. Ein interessantes Beispiel hierfür ist das Programm *rtftohtml*, das ab der aktuellen Version 3 nur noch als Shareware und damit nicht in Form von Quellcode zu bekommen ist. Angesichts mehrerer, bei näherer Betrachtung wirklich haarsträubender Implementierungsfehler, würde ich hier in Zweifelsfalls immer die ältere Version vorziehen, die noch als Quellcode verfügbar ist, und die vorhandenen Probleme selbst beseitigen, als mich in diesem Punkt allein auf den Autor des Programms zu verlassen.

Im Nachhinein betrachtet, ist auch die Lösung im Bereich der Grafikumwandlung sicherlich noch nicht optimal. In der jetzigen Form muß für jeden einzelnen Umwandlungsvorgang – das heißt für jedes einzelne Bild – ein nicht gerade kurzes Programm wie *convert* (aus dem *ImageMagick* Paket) immer wieder neu gestartet werden. Auch wenn man die Programmlaufzeit betrachtet, nimmt die Umwandlung der Bilder in den meisten Fällen viel mehr Zeit in Anspruch als die Umwandlung des reinen Dokumenttexts in HTML. Hier ließe sich bereits viel bei der Laufzeit einsparen, wenn man die Anzahl der dafür gestarteten Prozesse deutlich reduzieren könnte.

Eine mögliche Lösung, die sich für dieses Problem anbieten würde, ist die direkte Verwendung der Funktionen aus der *libMagick*. Diese Bibliothek enthält die gesamte Funktionalität von *ImageMagick* in Form von C-Funktionen[1], die dann auch die Bearbeitung von beliebig vielen Bildern in einem einzigen Programmdurchlauf ermöglichen. Allerdings müßte hierfür ein erheblicher Teil der Grafikumwandlung in C implementiert werden, was bisher nicht notwendig gewesen ist.

Eine besonders elegante Alternative dazu wäre wohl die Implementierung mit Hilfe von *PerlMagick* (siehe dazu [Magick 98] und die Quellenangabe in Anhang A), das sich als Modul in einen Perl-Interpreter laden läßt und dann dort die entsprechenden Funktionen aus der *libMagick* innerhalb der Programmiersprache Perl zur Verfügung stellt. Damit ließe sich eine recht elegante und effiziente Implementierung sogar in Perl realisieren.

Im Zusammenhang damit wäre aus heutiger Sicht sogar die Realisierung einer Plattform-unabhängigen grafischen Benutzeroberfläche in Perl denkbar, da in diesem Bereich inzwischen mit *Perl/TK* und *PerlGtk* schon einige recht leistungsfähige (aber nur eingeschränkt portable) Grafik-Toolkits für Perl zur Verfügung stehen, die ebenfalls das Modulkonzept von Perl verwenden. Hier wird sicherlich auch in der Zukunft noch mit weiteren interessanten Entwicklungen zu rechnen sein.

[1]Eigentlich ist *convert* nur eine Kommandozeilen-Schnittstelle für die *libMagick*.

Anhang A

Quellen

Dieser Anhang enthält noch einmal eine Zusammenfassung aller in dieser Diplomarbeit verwendeten Hilfsprogramme mit genauen Quellenangaben:

rtftohtml
Chris Hector <cjh@cray.com>
ftp://ftp.cso.uiuc.edu/pub/www/tools/translators/rtftoweb/

Perl
Larry Wall <larry@wall.org> und viele andere
ftp://ftp.perl.org/pub/perl/CPAN/

ImageMagick
John Cristy <magick@wizards.dupont.com> und viele andere
ftp://ftp.wizards.dupont.com/pub/ImageMagick/

PerlMagick
Kyle Shorter <magick@wizards.dupont.com>
ftp://ftp.wizards.dupont.com/pub/ImageMagick/perl/

Netpbm
Jef Poskanzer <jef@acme.com> und viele andere
ftp://ftp.cso.uiuc.edu/pub/X11/contrib/utilities/

Ghostscript
Aladdin Enterprises <ghost@aladdin.com>
ftp://ftp.cs.wisc.edu/ghost/gnu/

eps2tiff
Geoffrey Rutledge <rutledge@camis.stanford.edu>
ftp://ftp.leo.org/pub/comp/os/NEXTSTEP/graphics/convertors/

TIFF Utils
: Sam Leffler <sam@engr.sgi.com>
ftp://ftp.sgi.com/graphics/tiff/

tiff2png
: Willem van Schaik <gwillem@ntuvax.ntu.ac.sg>
ftp://ftp.nde.swri.edu/pub/png/applications/

OmniImageFilter
: Omni Development, Inc.
ftp://ftp.leo.org/pub/comp/os/NEXTSTEP/graphics/filter/

OmniWeb
: Omni Development, Inc.
ftp://ftp.leo.org/pub/comp/os/NEXTSTEP/connectivity/www/

Pager
: Axel T. Schreiner <axel@uos.de>
ftp://ftp.informatik.uni-osnabrueck.de/pub/axel/notes/

Ich habe meine Quellen zu *rtfdtohtml* abgelegt unter:

ftp://ftp.leo.org/pub/comp/os/NEXTSTEP/unix/text/

Anhang B

Bildformate

B.1 EPS – Encapsulated PostScript

Das EPS-Format ist aus der Grafik-Beschreibungssprache *PostScript* von *Adobe* hervorgegangen, die hauptsächlich für den Einsatz in Druckern entwickelt wurde, aber mit einem entsprechenden Interpreter (wie zum Beispiel Ghostscript) auch am Bildschirm betrachtet werden kann. *PostScript* ist eine mathematisch orientierte Vektorgrafiksprache, das heißt alle Bildelemente, mit Ausnahme von Bitmaps, werden als Zeichenoperationen in der Datei abgelegt. Damit läßt sich das Bild ohne Qualitätsverlust auf fast jede benötigte Größe und Druckerauflösung skalieren. EPS enthält nun neben der eigentlichen Grafikinformation auch noch Daten über die Position und Größe des Bilds, da man diese Informationen durch Analyse der Zeichenoperationen nur sehr schwer ermitteln kann.

Dieses Format wird typischerweise für die Darstellung von Text, Tabellen und vor allem Liniengrafiken verwendet, da sich solche Bilder besonders gut in Form von Vektorgrafik repräsentieren lassen. Das Einfügen von Bitmaps ist zwar möglich, diese werden dann aber nicht als Vektorgrafik gespeichert, was sich bei einer Skalierung durch entsprechend größere Pixel sehr nachteilig bemerkbar macht. Da EPS nicht komprimiert abgespeichert wird, sind Dateien in diesem Format im Vergleich zu den anderen Formaten im allgemeinen deutlich länger.

B.2 TIFF – Tagged Image File Format

Das TIFF-Format [TIFF 92] ist vor allem durch die Verwendung als Standardformat für Fax- und Scannergeräte bekannt geworden, wird aber darüber hinaus auch für andere Zwecke eingesetzt. Dieses Format ist – wie alle weiteren hier beschriebenen Bildformate – ein Bitmapgrafikformat, das Farbinformationen über jedes einzelne Pixel speichert. Eine Besonderheit hierbei ist, daß im TIFF-Format auch der Grad der Transparenz für jeden Bildpunkt mit abgespeichert werden kann, womit sich sehr ansprechende optische Effekte erzeugen lassen, wenn mehrere Bilder überlagert werden.

Ein weiterer Vorteil ist, daß eine TIFF-Datei mehrere Bilder enthalten kann. Diese Fähigkeit wird zum Beispiel benutzt, um mehrere Versionen eines Bilds in verschiedenen Farbtiefen in einer Datei abzulegen oder eine ganze Animation abzuspeichern. Das Komprimierungsverfahren läßt sich für jedes Bild einzeln festlegen, üblich sind dabei die Algorithmen *PackBits* und LZW[1] (*Lempel-Ziv & Welch*), sowie die CCITT-Verfahren zur Komprimierung von Fax-Dateien. Die Komprimierung erfolgt dabei verlustfrei, das heißt es geht keine Bildinformation verloren.

B.3 GIF – Graphics Interchange Format

Das von *CompuServe* entwickelte GIF-Format ist ebenso wie TIFF ein Bitmapgrafikformat, allerdings ist die Anzahl der Farben auf maximal 256 beschränkt, da eine Farbtabelle fester Größe verwendet wird. Außerdem ist im GIF-Format nur eine transparente Farbe vorgesehen (normalerweise die Hintergrundfarbe), so daß sich verschiedene Abstufungen bei der Transparenzeigenschaft nicht darstellen lassen.

GIF ist sicherlich das zur Zeit mit Abstand am weitesten verbreitete Bildformat, es wird zur Speicherung aller Arten von Bildern, Fotos, Icons und vielem anderen mehr verwendet. Für die Benutzung im *World Wide Web* wurde eine Variante[2] entwickelt, die bereits eine erste Bildvorschau erlaubt wenn erst ein kleiner Teil der Datei zur Verfügung steht. GIF kann ebenso wie TIFF mehrere Bilder in einer Datei ablegen, was vorwiegend für die bekannten GIF-Animationen benutzt wird.

Als Komprimierungsverfahren wird ausschließlich der verlustfreie LZW Algorithmus verwendet, der allerdings einem Patent der Firma *Unisys* unterliegt. Dieser Umstand war einer der Hauptgründe für die Entwicklung des neuen PNG-Formats, das in Abschnitt B.5 auf der nächsten Seite beschrieben ist. Auch in der TIFF-Spezifikation [TIFF 92] wird empfohlen, die LZW Komprimierung nicht mehr zu verwenden.

[1] Siehe auch den Hinweis dazu unter B.3.
[2] Das sogenannte *interlaced GIF* Format.

B.4 JPEG – Joint Photographic Experts Group

Speziell für die kompakte Speicherung von Fotos und ähnlichen Bildern mit vielen Farbverläufen, die sich mit den herkömmlichen Algorithmen nur schlecht komprimieren lassen, wurde von der *Joint Photographic Experts Group* das gleichnamige JPEG-Format [JPEG 93] entworfen. Es benutzt ein deutlich verbessertes Komprimierungsverfahren, das allerdings nicht mehr verlustfrei arbeitet, sondern die Bildinformation auf ein einstellbares Qualitätsniveau reduziert. Das Verfahren wurde dabei so gewählt, daß bei Foto-ähnlichen Bildern fast keine sichtbaren Veränderungen gegenüber dem Ausgangsbild auftreten, sich jedoch erhebliche Einsparungen im Platzbedarf erreichen lassen.

Durch die Spezialisierung auf diesen Bereich ist das Format allerdings für einfache Liniengrafiken sehr ungeeignet geworden, hier liefern EPS, GIF oder PNG deutlich bessere Resultate. Dennoch hat sich JPEG inzwischen als sehr beliebtes Format für Fotos durchgesetzt. Es gibt dabei keine Einschränkungen hinsichtlich der Anzahl der Farben, transparente Bilder wurden jedoch bei der Festlegung des Standards nicht vorgesehen. Auch hier gibt es die Variante *progressive JPEG*, die analog zu GIF eine schnelle Bildvorschau im WWW ermöglicht.

B.5 PNG – Portable Network Graphics

PNG ist ein relativ neues Format, das auf eine Initiative des *World Wide Web Consortiums* (W3C) entstanden ist [PNG 96]. Das Ziel dabei war, einen Ersatz für das GIF-Format zu finden, das wegen seiner zuvor bereits erwähnten Patentschwierigkeiten schon seit längerer Zeit sehr umstritten war. Es ist eine Weiterentwicklung des schon etwas älteren und heute nicht mehr gebräuchlichen IFF-Grafikformats. Als neue, verlustfreie Komprimierungsmethode wird der schon von den Programmen `zip` und `gzip` her bekannte und bewährte *deflate* Algorithmus verwendet, der im Vergleich zu GIF sogar im Durchschnitt um 10 bis 20 Prozent kürzere Dateien erzeugt.

Außerdem beseitigt dieses Format noch einige andere Schwachstellen, die sich beim GIF-Format negativ bemerkbar machen. Es gibt keine Einschränkungen bezüglich der Anzahl der verwendeten Farben, und zur Verbesserung der Darstellung wird der Wert für die Gammakorrektur zusammen mit dem Bild abgespeichert. Zudem kann PNG ebenso wie TIFF die komplette Transparenzinformation eines Bilds speichern. Für die Bildvorschau im *World Wide Web* wird hier ein verbessertes (zweidimensionales) Interlace-Schema verwendet.

Das PNG-Format erlaubt absichtlich nicht die Speicherung mehrerer Bilder in einer Datei. Für diesen Zweck gibt es ein erweitertes Format mit dem Namen *Multiple-Image Network Graphics* (MNG).

Literaturverzeichnis

[HTML 97] Dave Raggett: HTML 3.2 Reference Specification, W3C Recommendation Jan-1997, <http://www.w3.org/pub/WWW/MarkUp/>.

[RTF 87] Rich Text Format Specification. Microsoft Systems Journal, March 1987.

[Perl 96] Johan Vromans: Perl Programmers Reference Guide, January 1996, <http://www.perl.com/CPAN/>.

[TIFF 92] TIFF 6.0 Specification, <http://www.adobe.com/supportservice/devrelations/PDFS/TN/TIFF6.pdf>.

[JPEG 93] William B. Pennebaker, Joan L. Mitchell: JPEG Still Image Data Compression Standard, Van Nostrand Reinhold 1993.

[PNG 96] Portable Network Graphics Specification, W3C Recommendation Oct-1996, <http://www.w3.org/pub/WWW/Graphics/PNG/>.

[Manber 94] U. Manber, S. Wu: GLIMPSE: A Tool to Search Through Entire File Systems, Usenix Winter 1994 Technical Conference, San Francisco (January 1994), pp. 23-32.

[Magick 98] Kyle Shorter: An object-oriented Perl Interface to ImageMagick, <http://www.wizards.dupont.com/magick/www/perl.html>.

Erklärung

Hiermit erkläre ich gemäß §20 Absatz 6 der Diplomprüfungsordnung für den Studiengang Mathematik an der Universität Osnabrück, daß ich die hier vorliegende Diplomarbeit selbständig verfaßt und dafür keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Georgsmarienhütte, [Datum]