Institut für Informatik
der Technischen Universität München

Designspezifikationen im digitalen Publikationsprozess

Stefan Hermann

Institut für Informatik
der Technischen Universität München

Designspezifikationen im digitalen Publikationsprozess

Stefan Hermann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ. Prof. Dr. Dr.h.c. J. Eickel
Prüfer der Dissertation:	1.	Univ. Prof. Dr. A. Brüggemann-Klein
	2.	Univ. Prof. Dr. J. Schlichter

Die Dissertation wurde am 14.10.1999 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 4.7.2000 angenommen.

Danksagung

Zunächst einmal möchte ich mich bei Frau Prof. Dr. Anne Brüggemann-Klein bedanken, die mir den Weg in die Dokumentverarbeitung aufgezeigt und somit das Themengebiet meiner Dissertation ermöglicht hat. Herrn Prof. J. Schlichter gilt mein Dank für die Übernahme des Zweitgutachtens und das von ihm aufgebrachte Interesse an meiner Arbeit.

Herrn Prof. Derick Wood möchte ich an dieser Stelle zunächst für seine konstruktive Kritik an einigen Details meiner Arbeit danken, die mir rückblickend betrachtet mehr Kopfzerbrechen bereitet haben, als ich jeweils aktuell annahm. Insbesondere möchte ich Derick aber für sein Engagement und seine Unterstützung bei meinen Hong-Kong Aufenthalten danken.

All meinen Kollegen vom Lehrstuhl 11 danke ich für die herzliche Aufnahme und die Unterstützung in allen Phasen meiner Arbeit. Ohne die hier erhaltene Unterstützung wäre der Verlauf meiner Arbeit hier an der TU und auch meiner Dissertation sicherlich nicht so einfach gewesen, wie dies letztendlich der Fall war.

Auch möchte ich mich bei allen Studenten bedanken, die im Rahmen ihrer Studienarbeiten sowie durch anregende Diskussionen und Implementierungsarbeiten zum Erfolg der vorliegenden Arbeit beigetragen haben. Hervorheben darf und muss ich an dieser Stelle Herrn Christian Roth, ohne den meine Arbeit in wesentlichen Punkten nicht den heutigen Stand erreicht hätte.

Besonderer Dank gilt meiner Frau Michaela, die stets Verständnis für mein Unterfangen hatte, obwohl sie viele Abende und Wochenenden ohne mich verbringen musste. Meinem Sohn Dennis danke ich, dass er sich die letzten Tage vor seiner Geburt noch gemütlich Zeit gelassen hat, damit ich noch in Ruhe die letzten Zeilen meiner Arbeit fertigstellen konnte.

Schließlich möchte ich meinen Eltern für die Ausbildung danken, die sie mir ermöglicht haben. Ohne ihre fortwährende Unterstützung wäre diese Arbeit nie zustande gekommen.

Kurzfassung

Die Dokumentverarbeitung ist derzeit eines der wichtigsten Einsatzgebiete von digitalen Rechensystemen. 98% aller geschäftlich eingesetzten Rechensysteme werden zur Dokumentverarbeitung eingesetzt und 80% aller digital gespeicherten Informationen in Unternehmen liegen in Form von Dokumenten vor, Tendenz steigend. Durch den verstärkten Einsatz neuer Medien wird die Anzahl der täglich erstellten Dokumente immer größer, wobei zugleich die Anforderungen an deren Kundenanpassung, Wiederverwendung und Qualität immer höher gesetzt werden.

Der aktuelle Erstellungsprozess von Dokumenten ist dieser gewaltigen wirtschaftlichen und technischen Herausforderung nicht gewachsen. Einer Vielzahl von verfügbaren Software-Werkzeugen zur Dokumenterstellung steht ein enormes Methodendefizit gegenüber. Die Erstellung von Dokumenten erfolgt heute noch im Wesentlichen wie schon zu Beginn der rechnergestützten Dokumentverarbeitung, als Rechensysteme wie Schreibmaschinen benutzt wurden. Dokumente werden heute in der Regel mit Hilfe des Paradigmas des graphischen Auszeichnens inhärent als Einwegprodukte erstellt.

Als erster Schritt zur Lösung des soeben beschriebenen Missstandes wird schon seit den achtziger Jahren die logische Auszeichnung von Dokumenten vorgeschlagen. Sie ermöglicht die Trennung des eigentlichen Inhalts von Dokumenten von jeglicher Information, die zu ihrer Verarbeitung benötigt wird. Offen bleibt aber die Frage, in welcher Form die beabsichtigte Verarbeitung von logisch ausgezeichneten Dokumenten beschrieben werden kann. An dieser Stelle herrscht ein Methodendefizit, dessen systematische Behebung Ziel der vorliegenden Arbeit ist.

Die gegenwärtig für spezielle Anwendungsfälle verfügbaren ad hoc Lösungen dieses Methodendefizits liegen ausschließlich in Form von technisch motivierten Software-Umsetzungen vor. Im Gegensatz hierzu wird im Rahmen dieser Arbeit zunächst schrittweise eine Methodik (Designspezifikationsmethodik) zur Verarbeitung von logisch ausgezeichneten Dokumenten anhand einer systematischen Analyse des Bedarfs in der Industrie erarbeitet. Bei dieser Analyse wird nicht nur auf funktionale Aspekte geachtet, sondern im Hinblick auf die nötige Akzeptanz insbesondere auch die Personengruppe in den Mittelpunkt gerückt, die später mit der erarbeiteten Methodik arbeiten muss: Graphikdesigner, die für die Gestaltung der Produkte des Publikationsprozesses von Dokumenten verantwortlich sind. Sie sind Spezialisten im Bereich der Gestaltung und Typographie, nicht jedoch im Umgang mit komplexen Systemen, die ein intimes Wissen in Bezug auf die Programmierung von Rechensystemen erfordern.

Aufbauend auf der entwickelten Designspezifikationsmethodik wird anschließend eine vollständige Software-Architektur vorgestellt, auf deren Basis die Implementierung eines nutzerfreundlichen, graphischen Systems zur Bearbeitung von Designspezifikationen realisiert wurde. Von besonderem Interesse ist an diesem System insbesondere der Aufbau des Dokumentformatierers, der aufgrund seiner modularen Gestaltung konventionellen Dokumentformatierern überlegen ist und als Plattform für die weitere Verwendung in Forschung und Lehre geeignet ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Hintergrund

Die Dokumentverarbeitung ist zweifelsohne eines der wichtigsten Einsatzgebiete von digitalen Rechensystemen. Für viele dieser Systeme stellt sie sogar die wesentliche Daseinsberechtigung dar [7]. Untermauert werden diese Aussagen durch eine Analyse des Marktforschungsinstitutes Dataquest [46], wonach 98% aller geschäftlichen Computeranwender ihren PC zur Dokumentverarbeitung einsetzen. Beachtet man weiterhin, dass nach dieser Quelle 80% aller digital gespeicherten Informationen eines Unternehmens in Form von Dokumenten vorliegt, erkennt man unmittelbar die enorme wirtschaftliche Bedeutung der Dokumentverarbeitung. Demgemäß wird auch in [12] festgehalten, dass die Bundesrepublik Deutschland mit raschen Schritten auf dem Weg weg von der Industrie- hin zur Wissensgesellschaft ist. Informationen und die in ihrem Lebenszyklus stattfindenden Bearbeitungsschritte sind wirtschaftlich und gesellschaftlich von wesentlicher Bedeutung.

Durch den verstärkten Einsatz neuer Medien wird die Anzahl der täglich erstellten Dokumente immer größer, wobei zugleich die Anforderungen an deren Kundenanpassung und Qualität immer höher gesetzt werden. Fakten, die eine Änderung des aktuellen Erstellungsprozesses von Dokumenten, bei dem heute die reine Information in Dokumenten jeweils anwendungsspezifisch mit Verarbeitungsinformation „verschmutzt“ wird, dringlich fordern.

Ein typisches Beispiel für diesen Vorgang der Verschmutzung bieten die konventionellen Textverarbeitungssysteme, die wie Microsoft Word nach dem Paradigma des graphischen Auszeichnens arbeiten: Dokumente bestehen hier aus einer Sequenz von Text und eingestreuten Anweisungen, wie der Text graphisch für *einen* bestimmten Anwendungsfall aufbereitet werden soll.

Abhilfe schafft hier nur das Prinzip der Datenunabhängigkeit: Information wird in nichtproprietären, genormten Formaten gespeichert. Die Repräsentation gespeicherter Information ist vollkommen frei von jeglichen Artefakten, die zu ihrer konkreten Bearbeitung nötig sind.

Datenbanken, die Information strikt auf abstraktem Niveau und vollkommen losgelöst von Anwendungen verwalten, erzielten diese Datenunabhängigkeit bereits vor etlichen Jahren. Dokumente hingegen werden bis heute im Kontext einer konkreten Aufgabenstellung inhaltlich erstellt und gleichzeitig gelayoutet – und das auch noch mit proprietären Systemen.

Die so entstehenden Dokumente sind in mehrfacher Weise nicht zufriedenstellend: Zum einen sind sie technisch gesehen ein Wegwerfprodukt, eine Weiterverarbeitung oder Wiederverwendung (z.B. in abgewandelter oder neu kombinierter Form) ist nicht möglich. Dies wiegt um so schwerer, als die aufkommende Forderung, ein und dieselbe Information für verschiedenste Ausgabemedien, Vertriebskanäle und in mehreren Layouts – und in allen Punkten auch noch benutzerangepasst – anzubieten, nicht erfüllt werden kann. Zum anderen wird die Gelegenheit vertan, Information in Dokumenten reichhaltiger zu strukturieren, als dies im Vergleich zu Datenbanken möglich ist.

Lösungsideen für diesen Problembereich gibt es seit geraumer Zeit, wovon eine bereits im Jahre 1986 als internationale Norm ISO 8879 [32] verabschiedet wurde: SGML. Ihr Kerngedanke ist die strikte Trennung von Information, Struktur und Layout von Dokumenten, die durch die „reine“ logische Auszeichnung von Dokumenten erzielt wird. Der Wert dieser Idee ist von einigen bedeutenden Industriezweigen (z.B. der Luftfahrt- und Automobilindustrie) schon vor langer Zeit erkannt worden und wird sogar von einigen Behörden (z.B. dem amerikanischen DOD) seit etlichen Jahren als Standard-Dokumentformat gefordert.

Als erfolgreichstes und weltweit bekanntestes Beispiel einer SGML-Anwendung, das seinen Siegeszug in atemberaubender Geschwindigkeit hinter sich gebracht hat, ist das WWW anzuführen, das mit seiner Auszeichnungssprache HTML Erstaunliches geleistet hat.

Zur weiteren Propagierung von logisch ausgezeichneten Dokumenten hat das WWW-Konsortium[1] in letzter Zeit in Kooperation mit einer großen Anzahl von führenden Softwareherstellern den XML Standard [56] entwickelt, der nach heutigen Gesichtspunkten eine praxisnahe Vereinfachung des unnötig komplizierten SGML Standards darstellt. Mit seiner Hilfe soll es auf breiter Ebene ermöglicht werden, an spezielle Aufgabenstellungen angepasste logische Auszeichnungssprachen zu entwickeln, für die die populäre Auszeichnungssprache HTML aufgrund seiner engen Ausrichtung auf inhaltlich *einfache* WWW-Dokumente nicht ausreichend ist. Der Einsatz dieser an spezielle Aufgabenstellungen angepassten logischen Auszeichnungssprachen soll von der Datenseite her die oben genannten technischen Probleme der bisherigen Dokumentverarbeitung überkommen.

1.2 Problembeschreibung und Aufgabenstellung

Akzeptiert man die Feststellung, dass die logische Strukturierung von Dokumenten auf der Datenseite die grundlegende Voraussetzung für den Weg in die Wissensgesellschaft ist, stellt sich unmittelbar die Frage nach geeigneten Ansätzen und Systemen, mit deren Hilfe logisch ausgezeichnete Dokumente verarbeitet werden können: Logisch ausgezeichnete Dokumente

[1] Homepage auf www.w3.org.

enthalten keinerlei Vorschriften mehr, wie die in ihnen enthaltenen Informationen verarbeitet werden sollen.

Folgende in der Einleitung von [42] zu findende Aussage charakterisiert den momentanen Stand hierzu in der Praxis als auch in der Forschung sehr treffend:

> *The current state of electronic document processing can be characterized as a plethora of tools without a clear articulation of unifying principles and concepts underlying them.*

In obiger Aussage wird zwar einerseits festgehalten, dass es im Bereich der digitalen Dokumentverarbeitung eine Unmenge an Werkzeugen gibt, andererseits wird aber zugleich betont, dass trotz der Bedeutung der digitalen Dokumentverarbeitung dort bis heute ein Methodendefizit anzutreffen ist.

Dies trifft im Besonderen auf die Systeme für logisch ausgezeichnete Dokumente zu. Hier hat man sich zwar auf die Trennung von Information, Struktur und Layout von Dokumenten verständigt. Die verfügbaren Systeme sind aber proprietär, starr und lösen *lediglich* fest vorgegebene Aufgabenstellungen. Die Systeme tragen weiterhin meist den Nachteil in sich, speziell für wenige spezifische Klassen von logisch ausgezeichneten Dokumenten entwickelt worden zu sein. Sie sind vom methodischen Standpunkt aus betrachtet auf der gleichen Ebene wie Systeme, die nicht auf logisch ausgezeichneten Dokumenten aufsetzen und deshalb nicht den aufkommenden modernen Anforderungen (z.B. *Multiple Product Publishing*) gewachsen, die eine mehrfache automatische Verarbeitung von Dokumenten bedingen.

Weiterhin wird bei verfügbaren Systemen keinerlei Rücksicht auf die sie benutzenden Personengruppen sowie ihre persönliche und technische Qualifizierung genommen.

Die vorliegende Arbeit soll deshalb einen Beitrag zur Behebung des allgemeinen Methodendefizits im Bereich der digitalen Verarbeitung von logisch ausgezeichneten Dokumenten leisten.

Als das grundsätzliche Einsatzgebiet für digitale Verarbeitungssysteme von logisch ausgezeichneten Dokumenten, mit dessen Hilfe in dieser Arbeit eine genaue Analyse der bestehenden Defizite durchgeführt werden soll, wird der Publikationsprozess in Verlagen gewählt. Dieser stößt aufgrund der mangelnden Verfügbarkeit geeigneter Methoden und Systeme unter technischen und Kostenaspekten derzeit an die Grenzen seiner Leistungsfähigkeit (siehe hierzu auch [49]). Weiterhin verkörpert er mit den in ihm auftretenden Aufgabenstellungen prototypisch viele weitere Einsatzgebiete für digitale Dokumentverarbeitungssysteme.

Die Aufgabenstellung für diese Arbeit ist die Erstellung einer Methodik (die auch Designspezifikationsmethodik genannt wird), mit deren Hilfe die im Rahmen eines digitalen Publikationsprozesses für logisch ausgezeichnete Dokumente anfallenden maschinell ausführbaren Verarbeitungsschritte gesteuert werden können. Es gilt dabei zu beachten, dass im heutigen Publikationsprozess möglichst schnell und kostengünstig aufgrund eines Dokumentes mehrere Produkte in mehreren Zielmedien erstellt werden können müssen – und das wenn möglich automatisch und noch dazu angepasst an einen individuellen Kunden. Die Schlagworte hierfür sind *Multiple Product Publishing*, *Cross-Media Publishing* sowie *User Customized Publishing*.

Ergänzt wird die Aufgabenstellung durch die Forderung, dass die Ersteller von Designspezifikationen und ihre Qualifikation hierbei besonders berücksichtigt werden müssen: Sie sind Typographen und Graphikdesigner und somit Spezialisten im Bereich der Gestaltung und Typographie, nicht jedoch im Umgang mit komplexen Systemen, die ein intimes Wissen in Bezug auf die Programmierung von Rechensystemen erfordern.

Für eine Darlegung der Eignung des in dieser Arbeit gewählten methodischen Ansatzes gilt es weiterhin, für die vorgeschlagene Designspezifikationsmethodik eine softwaretechnisch umsetzbare Architektur zu entwickeln und mit ihrer prototypischen Realisierung die Funktionsfähigkeit des gesamten Systems zu beweisen.

1.3 Lösungsweg

Im Rahmen dieser Arbeit wird für obenstehende Aufgabenstellung der folgende Lösungsweg gewählt:

Aufgrund einer Analyse der automatisierbaren Anteile des Publikationsprozesses in Verlagen wird eine Designspezifikationsmethodik mit Namen [em:][2] entwickelt, die auf umfassende Weise die Kern- und erweiterten Aufgaben von Dokumentformatierern steuern kann. Sie überkommt somit das momentan in diesem Bereich bestehende Methodendefizit und bildet die Ausgangssituation für die weitere Entwicklung eines nutzerfreundlichen Systems [es:], mit dessen Hilfe ein rein digitaler Publikationsprozess unterstützt werden kann.

Funktional betrachtet besteht das System [es:] neben der Designspezifikationsmethodik [em:] zunächst aus einem Designprozessor [pe:], mit dessen Hilfe Designspezifikationen, die die Verarbeitung von logisch ausgezeichneten Dokumenten festlegen, automatisch auf beliebige Dokumente einer bestimmten Klasse (z.B. alle Briefe) angewendet werden können. Das Ergebnis eines derartigen Prozesses ist eine sogenannte Designstruktur, die basierend auf einem konsequent entwickelten objektorientierten Konzept auf neuartige Weise statisch einen Formatierprozess spezifiziert. Dieser Formatierprozess kann mit Hilfe eines für das System [es:] eigens entwickelten modularen Dokumentformatierers ausgeführt werden und liefert als Ergebnis eine Publikation, die gedruckt oder mit einem in den Dokumentformatierer [ef:] integrierten Betrachter angezeigt werden kann.

Wäre nun lediglich das Werkzeug [pe:] gegeben, so müssten die Ersteller von [em:] Designspezifikationen diese von Hand erstellen – ein Ansatz, der ausdrücklich in der Aufgabenstellung zu dieser Arbeit ausgeschlossen wurde: Das System soll die Ersteller von Designspezifikationen bezüglich ihrer Qualifikation geeignet unterstützen. Aus diesem Grund wird in [es:] ein interaktiv graphischer Designeditor [de:] zur Erstellung von Designspezifikationen verwendet, der auf dem Paradigma *Design by Example* basiert: Designspezifikationen werden mit Hilfe einer graphischen Umsetzung von [em:] in [de:] basierend auf konkreten Beispieldokumenten erstellt.

[2] Viele im Rahmen dieser Arbeit entwickelte Konzepte und Software-Module tragen einen Namen, der in lautschriftähnlicher Darstellung visualisiert wird.

Im Rahmen dieser Erstellung einer Designspezifikation mit Hilfe des Paradigmas *Design by Example* wird es den Nutzern von [de:] nicht nur ermöglicht, einzelne isolierte Designregeln zu erstellen, sondern im Rahmen eines ganzheitlichen Prozesses eine gesamte Designspezifikation als eine Einheit zu erarbeiten. Ein Ansatz, der es den Erstellern zum einen ermöglicht, wechselseitige Abhängigkeiten und Auswirkungen von Teilen der Designspezifikation einfach zu verstehen. Zum anderen wird aber auch die Übernahme von bestehenden Designspezifikationen mit diesem Ansatz vereinfacht.

Wie in allen produktiven Prozessen ist es auch nach der Erstellung einer Designspezifikation nötig, sie einer Kontrolle zu unterziehen. Aus diesem Grund wird der Designeditor [de:] zur Abrundung des Systems [es:] noch mit vielfältigen Möglichkeiten zum systematischen Testen und *Debuggen* von Designspezifikationen versehen.

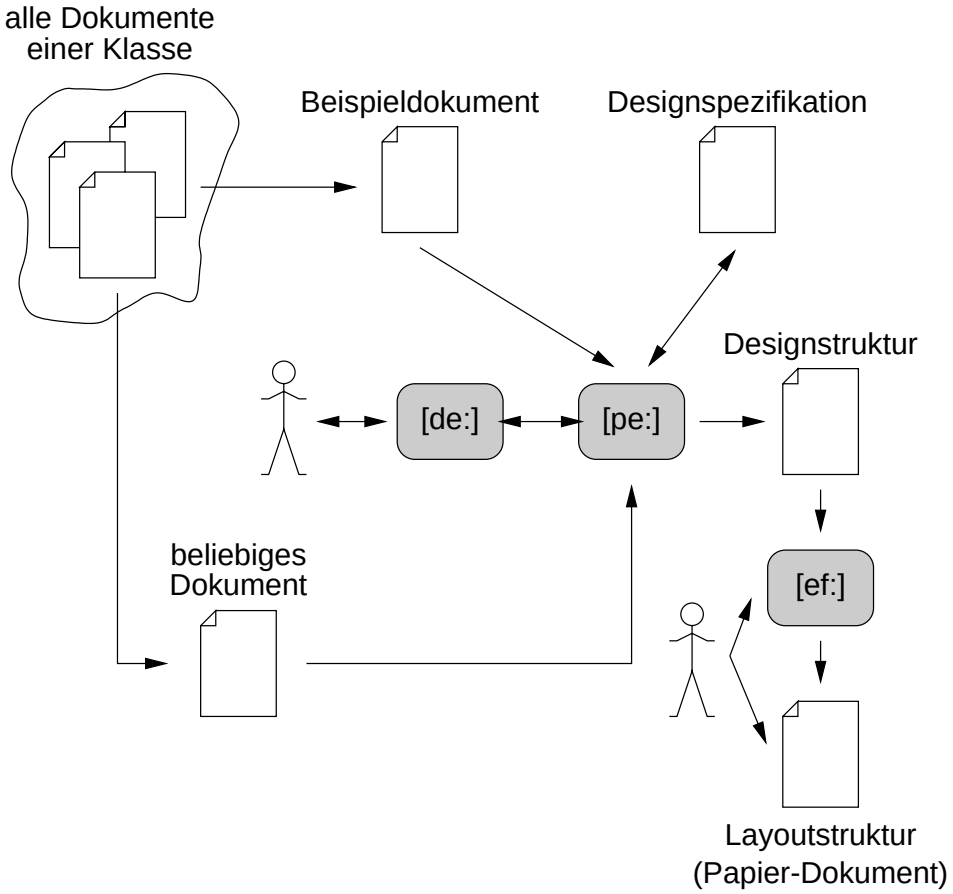

Abbildung 1.1: Grobarchitektur des [es:] Systems: Aus der Dokumentklasse, für die eine Designspezifikation erstellt werden soll, wird ein repräsentatives Dokument ausgewählt. Basierend auf dem gewählten Beispieldokument erstellt ein Graphikdesigner exemplarisch mit Hilfe des graphischen Werkzeuges [de:] die gewünschte Designspezifikation. Die dabei zur Ausführung von Berechnungen verwendete Komponente [pe:] dient später weiterhin zur Übersetzung eines beliebigen Dokumentes der vorgegebenen Dokumentklasse in die Designstruktur, die mit Hilfe des Formatierers [ef:] formatiert und visualisiert werden kann.

1.4 Übersicht

Der weitere Aufbau dieser Arbeit gestaltet sich wie folgt:

In Kapitel 2 wird der aktuelle Stand der Technik im Bereich der digitalen Dokumentverarbeitung betrachtet und eine klare und eindeutige Nomenklatur für diese Arbeit eingeführt. Dies ist nötig, da aktuell in diesem Forschungsbereich eine Vielzahl von Begriffen nur ungenau oder mehrfach unterschiedlich definiert ist, respektive bisher noch nicht formalisiert wurde. Hierzu wird in mehreren Schritten vorgegangen: Zunächst werden Dokumente und der Informationsbegriff an sich betrachtet, bevor auf die verschiedenen Möglichkeiten der digitalen Erstellung von Dokumenten eingegangen wird. Im Anschluß daran werden mögliche Konzepte zur Modellierung von Dokumenten sowie Modelle für Dokumentformatierer und aktuelle Stylesheet Modelle zur Verarbeitungssteuerung von Dokumenten vorgestellt.

Kapitel 3 beschäftigt sich mit der softwarebasierten Unterstützung des Publikationsprozesses. Hierzu werden zunächst die unterstützbaren Aufgaben analysiert, die sich im Rahmen der automatischen digitalen Verarbeitung logisch ausgezeichneter Dokumente überhaupt ergeben. Im Anschluß daran wird anhand mehrerer reeller Szenarien untersucht, in welcher Form diese Möglichkeiten derzeit in existierenden Ansätzen gesteuert werden können. Eine Analyse der in diesem Kapitel gewonnenen Ergebnisse führt schließlich zu der Aufgabenstellung für diese Arbeit, der Erstellung eines Systems zur Unterstützung des digitalen Publikationsprozesses von logisch ausgezeichneten Dokumenten, das die analysierten Schwachpunkte heutiger Systeme ausmerzt, modernen Anforderungen gewachsen ist und ergonomisch gestaltet ist.

In Kapitel 4 wird anhand eines systematischen Analyseansatzes die objektorientierte Designspezifikationsmethodik [em:] entwickelt, die als Basis eines digitalen Systems zur Unterstützung des digitalen Publikationsprozesses dienen kann. Hierbei werden zunächst sogenannte Layoutobjekte vorgeschlagen, die in übersichtlicher Form die Anwendung von Formatierfunktionen auf die logischen Bestandteile von Dokumenten spezifizieren. Weiterhin wird eine Systematik zum Aufbau einer Basismenge von Layoutobjekten vorgeschlagen die es ermöglicht, mit Hilfe der in der Basismenge enthaltenen Layoutobjekte *beliebige* Verarbeitungen für logisch ausgezeichnete Dokumente zu spezifizieren. Als ein zentraler Punkt wird in diesem Kapitel noch die sogenannte Designstruktur vorgestellt, mit deren Hilfe auf Basis der Layoutobjekte statisch ein Formatierprozess beschrieben werden kann. Für einen einfachen Zugang der Nutzer von [em:] zu dieser Designstruktur wird schließlich noch die leicht verständliche Pipeline-Metapher eingeführt, die anschaulich in die Bedeutung einer Designstruktur einführt.

Kapitel 5 behandelt die kontextabhängige Verarbeitung von Dokumenten. Hierzu werden zunächst die verschiedenen möglichen Kontext-Arten, die im Rahmen der Verarbeitung eines Dokumentes auftreten können, klassifiziert. Im Anschluß daran werden die für diese Arbeit besonders relevanten strukturellen Kontexte, die sich aus der logischen Struktur eines Dokumentes ergeben, behandelt. Zur Behandlung von strukturellen Kontexten in der Designspezifikationsmethodik [em:] wird weiterhin das T-Kontextmodell formal eingeführt und auf die graphische Spezifikation von T-Kontexten eingegangen.

In Kapitel 6 werden die Werkzeuge [pe:] und [de:] vorgestellt, mit deren Hilfe Designspezifikationen erstellt ([de:]) und auf logisch ausgezeichnete Dokumente angewendet werden

können ([pe:]). Der Schwerpunkt in diesem Kapitel liegt dabei auf [de:] und der graphischen Umsetzung der in Kapitel 4 eingeführten Designspezifikationsmethodik [em:] in eine graphische Benutzungsoberfläche, bei der besonderer Wert auf softwareergonomische Aspekte gelegt wird.

Kapitel 7 befasst sich mit der Entwicklung und Beschreibung des Dokumentformatierers und Betrachters [ef:], mit dessen Hilfe Designstrukturen ausgewertet werden können. Es wird die schlanke Architektur von [ef:] in Form eines Frameworks und beliebig ergänzbaren Layoutobjekten eingeführt, die im Rahmen mehrerer exakt definierter und vorgestellter Prozessabläufe zur Formatierung, Anzeige und Benutzerinteraktion miteinander interagieren können. Abschließend wird in diesem Kapitel schlüssig dargelegt, dass die für [ef:] gewählte Architektur den monolithischen Dokumentformatierern, wie sie momentan verfügbar sind, durch die einfache und übersichtliche Wartbarkeit und Erweiterbarkeit, sowie die inhärente Cross-Media Fähigkeit, deutlich überlegen ist.

In Kapitel 8 werden abschließend in kompakter Form die Ergebnisse der vorliegenden Arbeit zusammengefaßt. Es wird aufgezeigt, welche Ziele mit dieser Arbeit erreicht wurden und welchen Beitrag diese Arbeit zur wissenschaftlichen Forschung leisten kann. Abschließend wird noch ein kurzer Ausblick auf mögliche weiterführende Arbeiten gegeben, die im Zusammenhang mit den in dieser Arbeit realisierten Konzepten und der dafür entwickelten Software erfolgen können.

Kapitel 2

Erstellung und Bearbeitung digitaler Dokumente

2.1 Dokumente

Die vorliegende Arbeit befaßt sich mit Teilaspekten der rechnergestützten Erstellung und Bearbeitung von digitalen Dokumenten. Somit gilt es zunächst zu klären, was in dieser Arbeit unter dem Begriff Dokument und weiteren eng damit verknüpften Begriffen verstanden wird. Dieses Unterfangen mag auf den ersten Blick merkwürdig erscheinen, da Dokumente in unserem täglichen Leben allgegenwärtig sind. In der folgenden Betrachtung wird sich jedoch zeigen, dass sowohl der Begriff Dokument wie auch andere uns scheinbar klare Begriffe je nach Anwendergruppe gänzlich verschieden interpretiert werden und somit exakte Begriffsbildungen nötig sind.

2.1.1 Definitorische Ansätze

Erläuterungen und Definitionen zu dem Begriff Dokument finden sich gemäß seiner Bedeutung in den verschiedensten Quellen. So wird in [28] ein Dokument als eine „benannte, strukturierte Einheit von Text und möglicherweise Bildern, die gespeichert, bearbeitet, aufgesucht und zwischen Systemen oder Benutzern als selbständige Einheit ausgetauscht werden kann“, definiert. Eine gemäß den Erwartungen an eine Norm sehr technikzentrierte Definition, die nicht nur das Wesen von Dokumenten an sich festlegt („strukturierte Einheit von Text und möglicherweise Bildern“), sondern zugleich auch deren typische Verwendung in aktuell verfügbaren Systemen beschreibt.

Die in obiger Definition getroffene einschränkende Auffassung, dass ein Dokument im Wesentlichen ein *Schriftstück* ist, findet sich auch in [21]. Hier wird ein Dokument jedoch zusätzlich als „das zur Belehrung über etwas bzw. zur Erhellung von etwas dienliche“ charakterisiert, und somit die zentrale Eigenschaft von Dokumenten, nämlich Informationsträger zu sein, hervorgehoben. Dieser Aspekt der Informationsvermittlung als zentrale Aufgabe von Dokumenten findet sich auch in [32] („a collection of information“) sowie in weiterer aktueller Literatur.

Der Betonung des Aspektes der Informationsvermittlung folgt auch [9]. Hier wird die Auffassung vertreten, dass Dokumente beispielsweise Träger von Text, Literatur, geschriebener Sprache, organisierten Gedankengängen und somit insgesamt Repräsentanten von Wissen sind, ohne dass dabei ein bestimmtes Speicher- oder Präsentationsmedium mit dem Dokument verknüpft wird.

Im Gegensatz zu dieser Körperlosigkeit von Dokumenten in [9] wird in [50] zur Betonung der mannigfaltigen Möglichkeiten der physikalischen Existenz von Dokumenten schließlich darauf hingewiesen, dass Dokumente nicht nur als gedruckte Schriftstücke, sondern in beliebiger Form (z.B. als Bild, Video oder Sprachaufzeichnung) vorliegen können, eine Tatsache, die in der heutigen Zeit aufgrund der rasanten Entwicklung der Multimediafähigkeit von Rechensystemen immer mehr an Bedeutung gewinnt.

Resümiert man die genannten Quellen, ist ein Dokument ein komplexeres Gebilde als dies im alltäglichen Sprachgebrauch erscheinen mag. Faßt man obige Ansichten zu einer einzigen zusammen, ist ein Dokument zunächst ein physikalisch greifbarer Gegenstand (z.B. Schriftstück, Bild, Video, Sprachaufzeichnung), über den Nutzer zumindest für eine gewisse Zeit verfügen können. Mit Hilfe der diesen Gegenstand vermittelnden Informationsträger (auch Repräsentationen genannt), wie z.B. Text oder Sprache, haben die Nutzer Zugriff auf die eigentlichen Informationen des Dokumentes, die gegebenenfalls untereinander in gewissen Beziehungen stehen können.

Diese Charakterisierung eines Dokumentes gebe ich in folgender Definition wieder:

Definition 1 (Dokument) *Ein Dokument besteht aus einer zumindest kurzfristig gespeicherten Menge von strukturierten Daten, die mit Hilfe von Repräsentationen präsentiert werden.*

Zusammen bilden die *Daten* und deren *Struktur* die in einem Dokument enthaltene *Information*. Unter Information verstehen wir den abstrakten Gehalt („Bedeutungsinhalt“, „Semantik“) eines Dokumentes. Die *Struktur* beschreibt die Beziehung der Daten untereinander. Elemente der Struktur sind in einem typischen Dokument z.B. Kapitel, Überschriften, Absätze, Aufzählungen und Verweise.

Die *Repräsentation* bestimmt die sinnlich wahrnehmbare Erscheinung eines Dokumentes. Üblicherweise werden in Dokumenten Daten visuell in Form von Text repräsentiert, wobei die Elemente der Struktur beispielsweise durch die Variierung von Schriftattributen und unterschiedliche Abstandssetzung ausgezeichnet werden. Für blinde Personen besteht die Möglichkeit, als Repräsentation für die Daten Braille-Zeichen zu verwenden. Alternativ hierzu ist auch denkbar, Sprache als Repräsentation einzusetzen. Die *Präsentation* ermöglicht die sensorische Wahrnehmung von Repräsentationen durch die Nutzer eines Dokumentes. Möglichkeiten hierzu sind z.B. der Druck eines Dokumentes auf Papier, die Anzeige auf einem Bildschirm, das Stanzen von Braille-Zeichen in Papier, oder das Wiedergeben von Sprache über ein geeignetes System.

Die Forderung der zumindest kurzfristigen Speicherung bewirkt, dass beispielsweise die Inhalte von Gesprächen, bei denen zwar strukturierte Informationen ausgetauscht, diese aber nicht aufgezeichnet werden, nicht als Dokumente betrachtet werden können.

Ein Dokument im Sinne obiger Definition ist durch seine Präsentation der Wahrnehmung durch den Menschen zugänglich und umfaßt nicht nicht nur schriftliche Aufzeichnungen (Buchstabenfolgen, die wir im Folgenden als Text-Dokumente bezeichnen), sondern auch z.B. Bilder und Videos. Im weiteren Verlauf dieser Arbeit beschränken wir uns jedoch in Beispielen, wenn nicht anders vermerkt, der Übersichtlichkeit halber auf Text-Dokumente.

Bei genauer Analyse von Definition 1 können Dokumente unter verschiedenen Blickwinkeln betrachtet werden. Zwei wichtige Blickwinkel fokussieren dabei zum einen auf die in einem Dokument enthaltene Information und zum anderen auf die äußere Form eines Dokumentes. Zum vereinfachten Sprachgebrauch in dieser Arbeit werden deshalb zwei weitere Definitionen eingeführt, mit deren Hilfe der Schwerpunkt der Betrachtung bei der Untersuchung bestimmter Aspekte eines Dokumentes deutlich gekennzeichnet werden kann.

Definition 2 (Abstraktes Dokument) *Mit der Verwendung des Begriffes abstraktes Dokument wird gekennzeichnet, dass man bei der Betrachtung eines Dokumentes speziell Bezug auf dessen Struktur und Daten*[1] *nimmt. Die äußere Form des Dokumentes spielt in diesem Fall keine Rolle.*

Definition 3 (Konkretes Dokument) *Bei Verwendung des Begriffes konkretes Dokument wird betont, dass bei der Betrachtung eines Dokumentes die den Nutzern des Dokumentes direkt zugängliche physikalische Präsentation des Dokumentes im Vordergrund steht.*

Aufgrund der Aufgabenstellung dieser Arbeit gilt es nun noch den Begriff *digitales Dokument* zu definieren. Dies gestaltet sich mindestens ebenso schwierig, wie es bei dem Begriff Dokument der Fall war. Während der Begriff Dokument ein allgemeines Konzept beschreibt, das den meisten Menschen in groben Zügen ohne weitere Erklärung intuitiv klar ist, beschreibt der Begriff digitales Dokument ein neues Artefakt, dessen Bedeutung noch sehr frei aufgefaßt werden kann.

Unterschiedliche Faktoren, die die Anforderungen an ein digitiales Dokument beeinflussen, und somit eine Definition des Begriffes digitales Dokument prägen können, ergeben sich insbesondere aus den verschiedenen Phasen des Lebenszyklus eines Dokumentes. Besonders hervorzuheben sind dabei zumindest die Phasen der Erstellung, Suche, Distribution und Nutzung. Auch die Weiterverwendung eines Dokumentes nach seinem eigentlichen geplanten Lebenszyklus ist von gravierender Bedeutung für sein digitales Gegenstück und dessen Definition.

Eine zu obigen Feststellungen konforme Ansicht vertitt der ODA Standard [29], indem hier die digitale Form, in der ODA-Dokumente gespeichert sind, je nach Anforderung verschieden definiert ist. Prinzipiell werden in ODA verschiedene Formen für die digitale Version des Dokumentes unterschieden, die sich zum einen durch die betroffene Lebenszyklusphase, und zum anderen durch eine gegebene weitere Bearbeitungseignung bzw. -nichteignung auszeichnen.

Der Begriff digitales Dokument findet derzeit weite Verbreitung in wissenschaftlichen Publikationen. Tatsache ist allerdings, dass er dabei nur verwendet wird, ohne durch explizite

[1] Wenn nicht anders vermerkt zielen folgende Verwendungen des Begriffes *abstraktes Dokument* in dieser Arbeit nicht auf dessen *aktuelle* Daten und deren Struktur ab. Es wird lediglich die Vernachlässigung äußerer Aspekte des Dokumentes betont.

Definitionen charakterisiert zu werden. Vielmehr ergibt sich die jeweilige Bedeutung implizit aus der allgemeinen Thematik, in der er verwendet wird.

Weiterhin interessant zu verfolgen ist die Tatsache, dass der Bestandteil *digital* erst in letzter Zeit das Wort *elektronisch* abgelöst hat. Dies gilt dabei nicht nur für den Begriff digitales Dokument, sondern auch für weitere Begriffe, die den Bestandteil digital beinhalten. Ein Beispiel hierfür ist der Begriff digitale Bibliothek, der sich aus dem Begriff elektronische Bibliothek heraus entwickelt hat. Auch der Titel der wichtigsten Tagungsreihe im Bereich der Dokumentverarbeitung wird sich demnächst ändern. War er seit der ersten Tagung im Jahr 1986 *Electronic Publishing*, so wird er ab dem Jahr 2000 *Digital Documents and Electronic Publishing* sein.

Bei der Festlegung der Definition zu digitalen Dokumente für diese Arbeit ist zu beachten, dass sich diese Arbeit mit der digitalen Verarbeitung von Dokumenten von deren Erstellung bis hin zur Betrachtung durch die Nutzer beschäftigt. In ihr werden somit annähernd alle Phasen des Lebenszyklus eines Dokumentes zumindest kurz angesprochen. Aus diesem Grund muss auch die hier zu treffende Definition für digitale Dokumente entsprechend flexibel vorgenommen werden. Eine naheliegende Möglichkeit hierzu ist die Aufgabe des Zieles, eine konkrete Definition zu erarbeiten, die technisch auf die betroffenen Phasen des Lebenszyklus eines Dokumentes eingeht. Statt dessen scheint es angezeigt eine abstrakte Definition zu erstellen, die losgelöst von den Phasen ein gleichwertiges Gegenstück zu der Definition eines Dokumentes darstellt. In folgender Definition wird dieser Weg gewählt:

Definition 4 (Digitales Dokument) *Ein digitales Dokument ist ein Dokument, das in einem digitalen Rechensystem gespeichert ist, und mit seiner Hilfe verarbeitet werden kann. Je nach Eignung der digitalen Speicherung und des Systems können unterschiedliche Bestandteile des digitalen Dokumentes verarbeitet werden.*

Anzumerken ist, dass wir in der Definition zu digitalen Dokumenten nicht festlegen, mit welchen konkreten Systemen ein Dokument bearbeitbar sein muss. Die unterschiedlichen Bestandteile eines Dokumentes, die prinipiell mit Hilfe eines Systems verarbeitet werden können, sind im folgenden Abschnitt genannt.

Komponenten von Dokumenten

Nach Definition 1 lassen sich Dokumente in die vier Komponenten *Daten*, *Struktur*, *Repräsentation* und *Präsentation* aufteilen (siehe Abbildung 2.1).

Übliches Ziel der Nutzer eines Dokumentes ist die Erfassung der im Dokument enthaltenen Daten und ihrer Struktur unter der Prämisse, möglichst die Daten in der Strukturierung zu erhalten, wie sie der Autor des Dokumentes vermitteln wollte (siehe Abbildung 2.2). Aufgabe der Präsentation eines Dokumentes ist es somit, die von einem Autor gegebenen Daten samt ihrer Strukturierung über die Hilfe von Repräsentationen den Nutzern zur Informationsaufnahme darzubieten. Dreh- und Angelpunkt einer reibungslosen Informationsübermittlung ist somit die geeignete Repräsentation der in einem Dokument präsentierten strukturierten Daten.

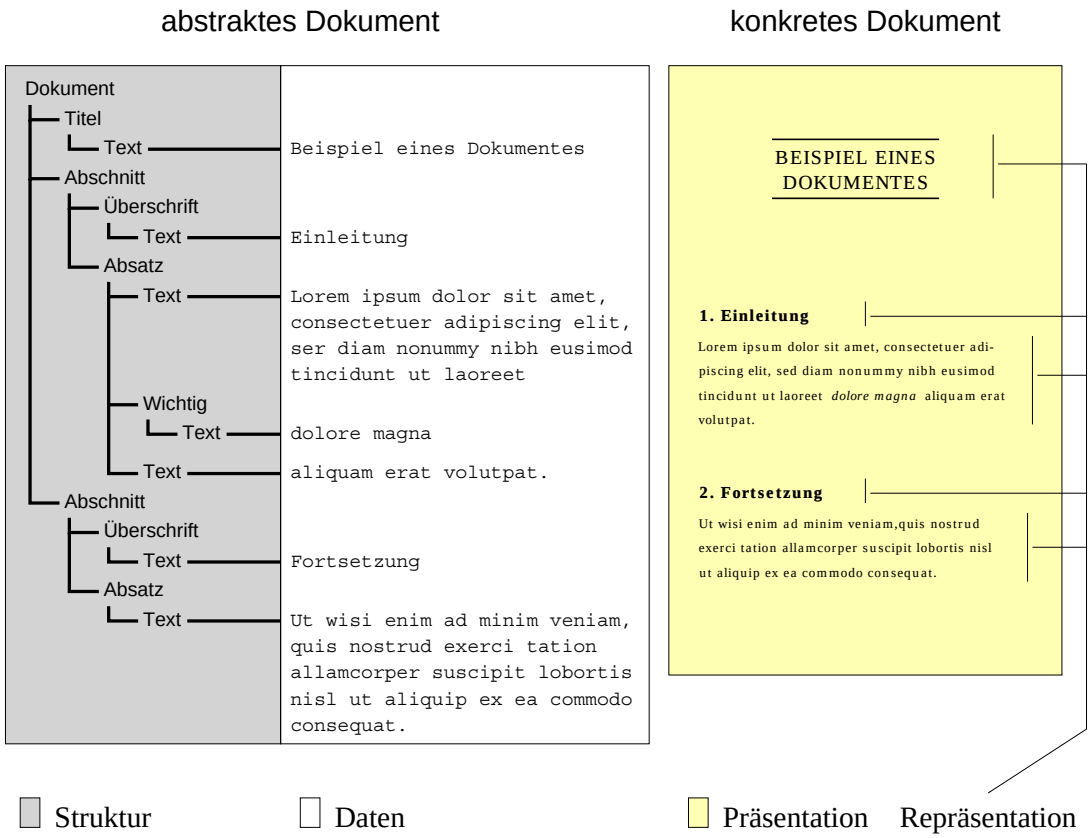

Abbildung 2.1: Komponenten eines Dokumentes: Auf der linken Seite der Abbildung sind die Daten und Struktur (abstraktes Dokument) eines einfachen Text-Dokumentes dargestellt. Auf der rechten Seite ist eine ausgewählte Präsentation der Repräsentationen (konkretes Dokument) dieses Text-Dokumentes zu sehen.

Die Repräsentation von Text-Dokumenten bestimmt sich im Allgemeinen zum einen durch die Wahl von visuellen Attributen (Schrift, Einrückung, Farbe, ...) für die verschiedenen Dokumentbestandteile und deren Anordnung in einer Präsentation sowie zum anderen durch die Zurverfügungstellung von generierten Verzeichnissen, Indexen und weiteren hier nicht genannten Bestandteilen. Als Beispiele hierzu werden nach allgemeinen Konventionen Überschriften meist in größerer Schrift als der normale Fließtext und zusätzlich abgesetzt von diesem gesetzt, Nummerierungen für Bestandteile wie z.B. Abbildungen, Tabellen und Überschriften eingeführt, Hervorhebungen durch Schriftwechsel angezeigt, Fußnoten getrennt vom Fließtext am unteren Ende der Seiten gesetzt und zu Beginn respektive am Ende des Dokumentes Inhaltsverzeichnisse und Indexe erzeugt. In konform zu obigen Beispielen gesetzten Dokumenten ist die Struktur des Dokumentes für die Nutzer und damit zugleich auch der Zugriff auf die im Dokument enthaltene Information einfacher möglich als bei unkonventionell gesetzten Dokumenten.

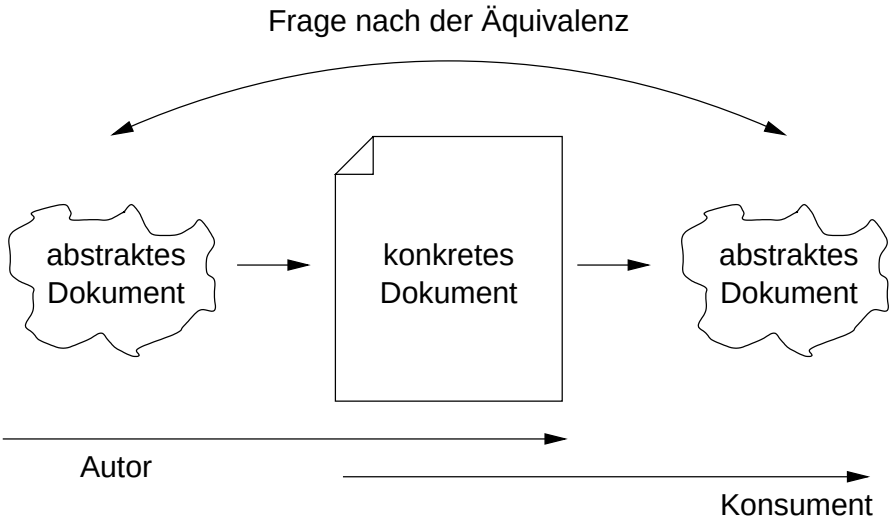

Abbildung 2.2: Informationsvermittlung vom Autor zum Nutzer: Das vom Autor intendierte abstrakte Dokument wird in Form eines konkreten Dokumentes als Schnittstelle den Nutzern angeboten. Aus diesem konkreten Dokument erstellen sich die Nutzer ihr eigenes abstraktes Dokument. Es stellt sich die Frage nach der Äquivalenz des aus dem konkreten Dokument rekonstruierten abstraktem Dokument und dessen abstraktem Ausgangsdokument.

2.1.2 Daten und Struktur

Nach Definition 1 sind die in Dokumenten enthaltenen Daten strukturiert. Wie wir mit den folgenden Definitionen zeigen werden, kann diese strukturelle Beziehung der Daten in Dokumenten zueinander aus verschiedenen Blickwinkeln heraus betrachtet werden.

Die Struktur eines Dokumentes wird umso wichtiger, je größer ein Dokument wird. Ohne eine geeignete Struktur oder gänzlich ohne Struktur ist es fast unmöglich, die Information eines Dokumentes aufzunehmen. Aus diesem Grund führt man in den meisten Dokumenten *hierarchische Strukturen* ein, die den Nutzern den Umgang mit komplexen Informationen vereinfachen. So kann ein Buch beispielsweise aus einem Titel und einer Folge von Kapiteln bestehen, die hierarchisch aus einer Folge von Abschnitten zusammengesetzt sind, welche wiederum Überschriften, Paragraphen und Listen beinhalten. Diese durch die Einführung von hierarchischen Strukturen entstehenden Einheiten nennen wir gemäß der folgenden Definition *logische Objekte*:

Definition 5 (Logische Objekte) *Die durch die Einführung einer hierarchischen Struktur in einem Dokument entstehenden Datenblöcke*[2] *nennen wir im Folgenden logische Objekte*[3]*. Um anzuzeigen, welche Aufgabe ein Datenblock in einem Dokument übernimmt, können logische Objekte mit einem Namen bezeichnet werden.*

[2] Z.B. Kapitel, Abschnitte und Paragraphen.

[3] Siehe auch [28]: Mit logischem Objekte werden hier all diejenigen Dokumentbestandteile bezeichnet, die für eine Anwendung oder einen Benutzer eine signifikante Bedeutung haben. Und [3]: Mit logischen Objekten werden all die Objekte bezeichnet, die man identifizieren kann, wenn man ein Dokument unter logischen Gesichtspunkten analysiert.

Definition 6 (Hierarchische/Logische Struktur) *Die hierarchische Struktur eines Dokumentes ergibt sich durch die rekursive Aufteilung der Daten des Dokumentes in logische Objekte. Die hierarchische Struktur wird auch mit dem Begriff logische Struktur bezeichnet.*

Es ist an dieser Stelle ausdrücklich zu betonen, dass nach obiger Definition die aus logischen Objekten zusammengesetzte hierarchische Struktur von Dokumenten (sofern man das Dokument an sich als Wurzel betrachtet) als Datenstruktur ein Baum ist.

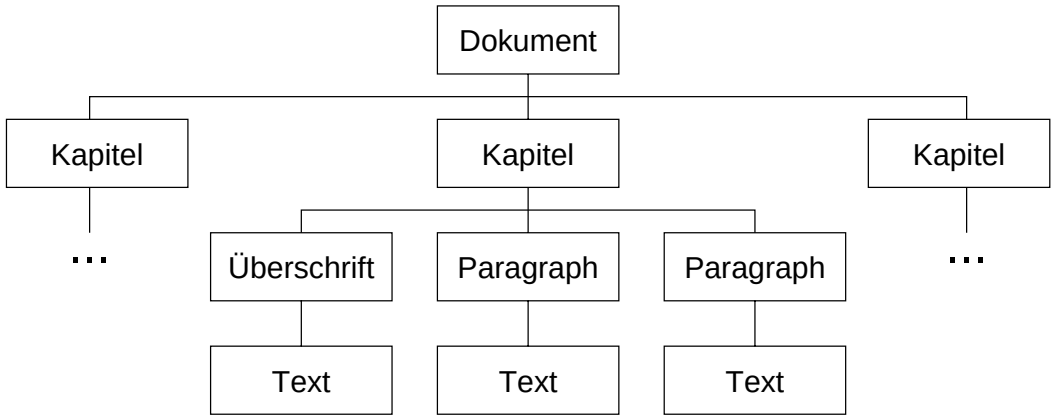

Abbildung 2.3: Hierarchische Struktur eines Dokumentes, das aus drei Kapiteln besteht. Das zweite Kapitel beinhaltet eine Überschrift und zwei Paragraphen, die alle Text beinhalten. Der Übersicht halber ist der Inhalt des ersten und dritten Kapitels nicht dargestellt.

Die Daten eines Dokumentes, die in Form von logischen Objekten in einer hierarchischen Struktur angeordnet sind, können von den Nutzern nur sequentiell konsumiert werden. In einem gewöhnlichen Text-Dokument gibt es weitgehend (unter Vernachlässigung von Fußnoten und ähnlichen Verweismöglichkeiten) nur eine einzige lineare Sequenz, in welcher Reihenfolge das Dokument gelesen wird: Zuerst wird die erste Seite von oben nach unten gelesen, dann die zweite, u.s.w. [44]. Die dabei entstehende flache Struktur der Abfolge der logischen Blatt-Objekte eines Dokumentes bezeichnen wir mit der folgenden Definition als linearisierte Struktur des Dokumentes.

Definition 7 (Linearisierte Struktur) *Die lineare Abfolge*[4] *der logischen Blatt-Objekte der hierarchischen Struktur eines Dokumentes bezeichnet man als seine linearisierte*[5] *Struktur.*

In der linearisierten Struktur eines Dokumentes kommen nur noch die terminalen logischen Blatt-Objekte eines Dokumentes wie z.B. Überschriften und Absätze vor. Die inneren logischen Objekte wie z.B. Kapitel sind in ihr nicht mehr enthalten.

Teilmengen der in Dokumenten beinhalteten logischen Objekte können neben der durch die hierarchische Struktur eingeführten Abhängigkeit untereinander noch in einer zweiten, nicht

[4] Z.B. durch einen depth-first Durchlauf erzielt.

[5] Sie stimmt in der Regel weitgehend mit der normalen linearen Lesereihenfolge eines Dokumentes überein.

notwendigerweise hierarchischen Art von Abhängigkeit stehen. Diese Abhängigkeit wird mit Hilfe von Verweisen (auch Referenzen genannt) zwischen den einzelnen Mitgliedern der Teilmengen aufgebaut. Typische Beispiele für Verweise in Dokumenten sind Querverweise, Literaturverweise und Indexe. Verweise sind funktional gerichtet, d.h. sie gehen von einer Quelle (dem Startpunkt für eine Aufgabe, wie z.B. dem Versuch der Verdeutlichung eines Sachverhaltes in Form einer Fußnote) aus und zeigen auf ein Ziel (z.B. einem Eintrag im Literaturverzeichnis).

Definition 8 (Verweis) *Ein Verweis setzt eine Teilmenge (Ausgangsmenge) der logischen Objekte eines Dokumentes mit einer nicht notwendigerweise disjunkten Teilmenge (Zielmenge) der logischen Objekte eines Dokumentes in eine gerichtete*[6] *Beziehung. Die Kardinalität sowohl der Ausgangs- als auch der Zielmenge kann 1 oder n sein.*

Eine Menge von Verweisen, die für ein Dokument gegeben ist, führt in diesem Dokument eine neue Struktur ein, die wir gemäß folgender Definition Verweisstruktur nennen:

Definition 9 (Verweisstruktur) *Die Verweisstruktur eines Dokumentes definiert sich über die Menge aller für das Dokument vorgegebenen Verweise.*

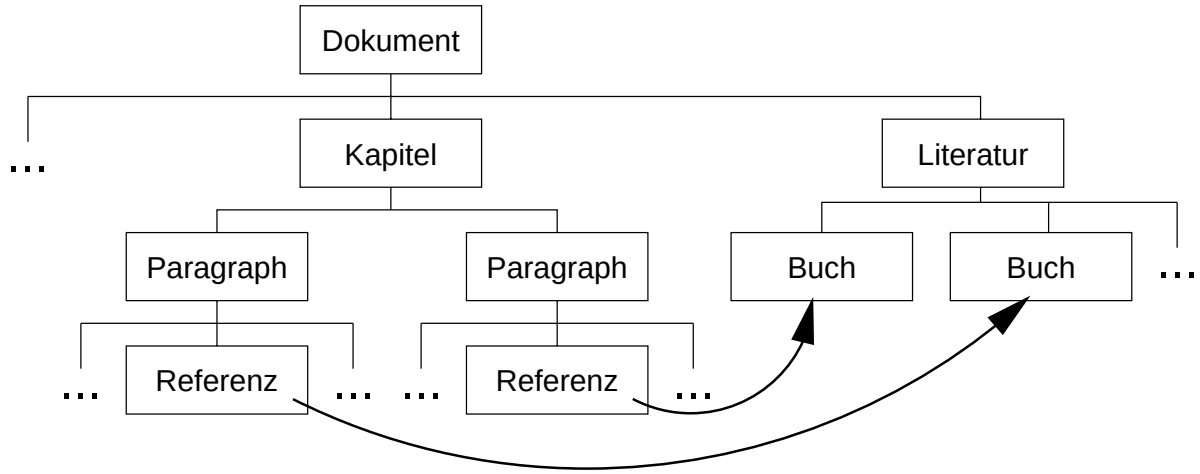

Abbildung 2.4: In der hier dargestellten Verweisstruktur eines Dokumentes wird von zwei Punkten des Textteils aus Bezug auf zwei verschiedene Bücher im Literaturverzeichnis genommen.

Die Verweisstruktur wird in der Literatur auch mit *nichtlinearer Struktur* [47, 50] bezeichnet. Sie reichert den durch die hierarchische Struktur vorgegebenen Strukturbaum eines Dokumentes um gerichtete Verweise an. Als Ergebnis dieser Anreicherung des Baumes ergibt sich als neue Datenstruktur für die Struktur des Dokumentes ein gerichteter Graph.

[6] Quelle und Ziel der Relation sind unterscheidbar.

Die Nutzung eines Dokumentes mit einer gegebenen Verweisstruktur kann durch die Nutzer in mehreren Varianten erfolgen. Bei der Lektüre eines Buches mit Fußnoten und Literaturreferenzen bestimmt jeder einzelne Leser individuell seinen eigenen Nutzungspfad, der für ihn individuell die linearisierte Struktur des ihm vorliegenden Dokumentes darstellt.

Anstelle eines einzigen vordefinierten Informationsflusses stellen die Autoren von Dokumenten mit einer gegebenen Verweisstruktur den Nutzern eine Reihe von mehreren verschiedenen Explorationsmöglichkeiten zur Verfügung. Die linearisierte Struktur von Dokumenten, die eine Verweisstruktur besitzen, ist deshalb mehrdeutig.

2.1.3 Repräsentation und Präsentation

In Definition 1 zu Dokumenten ist festgelegt, dass Daten und Struktur (logische Objekte) eines Dokumentes mit Hilfe von Repräsentationen den Nutzern präsentiert werden. Typische Repräsentationen[7], die dabei verwendet werden, sind Paragraphen, Buchstaben, u.s.w.. Zur Präsentation der oben aufgeführten Repräsentationselemente werden üblicherweise Papier und Bildschirm[8] verwendet. Zusammen bezeichnen wir die angeführten Repräsentations- und Präsentationselemente als *Layoutelemente*:

Definition 10 (Layoutelement) *Die zur Repräsentation und Präsentation der logischen Objekte eines Dokumentes verwendeten Elemente nennen wir Layoutelemente*[9].

Die hierarchische Zusammensetzung von Layoutelementen ergibt ein dem Menschen zugängliches konkretes Dokument. Üblicherweise ist dies eine Reihe von Seiten (Präsentationselement), wobei jede Seite von oben nach unten mit Layoutelementen wie Überschriften, Paragraphen, Listen, Bildern, u.s.w. (realisiert durch Repräsentationselemente) gefüllt wird. Die sich ergebende Struktur wird *Layoutstruktur* genannt:

Definition 11 (Layoutstruktur) *Die Layoutstruktur eines Dokumentes ist die statische hierarchische Struktur der Layoutelemente, die zur Repräsentation und Präsentation der in einem Dokument beinhalteten Daten samt ihrer Struktur verwendet werden.*

Die Layoutstruktur eines Dokumentes ist als Datenstruktur betrachtet ein Baum, der das *Layout* eines Dokumentes festlegt:

Definition 12 (Layout) *Unter dem Layout eines Dokumentes verstehen wir die den Nutzern zugängliche Erscheinungsform eines Dokumentes (das konkrete Dokument, siehe Definition 3).*

Es ist zu beachten, dass Layoutelemente z.B. in gedruckten Präsentationen umgebrochen und auf mehrere Seiten verteilt werden können. Die dabei entstehenden Fragmente der Layoutelemente nennen wir Layouteinheit:

[7] Im Folgenden Repräsentationselemente genannt.

[8] Im Folgenden Präsentationselemente genannt.

[9] Siehe auch [28]: Layoutelemente werden dort mit dem Begriff Layoutobjekt bezeichnet und beschreiben die Elemente der gestalterischen Struktur eines Dokumentes. Und [3]: Mit Layoutobjekt werden hier all die Objekte bezeichnet, die man beim Betrachten eines Dokumentes identifizieren kann.

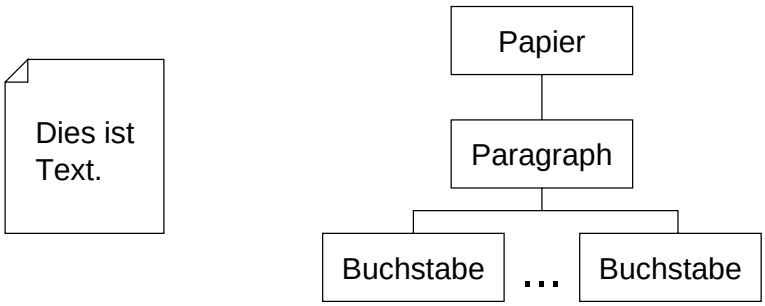

Abbildung 2.5: Auf der rechten Seite dieser Abbildung ist der Layoutstruktur-Baum des auf der linken Seite dargestellten konkreten Dokumentes abgebildet. An seiner Wurzel befindet sich zur Realisierung der Darstellung eine Präsentationseinheit Papier. Ihm untergeordnet ist ein Repräsentationselement Paragraph, das wiederum mehrere Repräsentationselemente vom Typ Buchstabe enthält.

Definition 13 (Layouteinheit) *Die bei der Aufteilung eines Layoutelementes entstehenden Fragmente werden Layouteinheiten genannt. Sie werden als gleichwertig zu Layoutelementen betrachtet.*

Zur Realisierung von Layoutelementen werden *Medien*[10] verwendet. Da der Begriff Medium insbesondere in der Informatik mehrere Bedeutungen hat, muss zum exakten Verständnis die Verwendung dieses Begriffes geklärt werden. Wir untersuchen dazu zunächst, wie der Begriff Medium in der Literatur klassischer Weise verstanden wird. Im Anschluss daran wenden wir uns denjenigen Aspekten dieses Begriffes zu, die erst durch die rechnergestützte Bearbeitung von Dokumenten möglich wurden.

Klassische Betrachtung

In [21] wird Medium als „vermittelndes Element und Kommunikationsmittel“ beschrieben, in [39] als „Mittel zur Weitergabe oder Verbreitung von Information durch Sprache, Gestik, Mimik, Schrift und Bild“. Diese allgemeinen Festlegungen sind von den gebräuchlichen Formen zwischenmenschlicher Kommunikation abgeleitet und spiegeln die umgangssprachliche Verwendung des Begriffes sehr gut wieder. Für unsere Zwecke sind sie jedoch nicht exakt genug. Die in [31] vorgenommene exakte Klassifizierung des Begriffes Medium nach verschiedenen technischen Kriterien, die insgesamt zu sieben verschiedenen Auffassungen des Begriffes Medium führt, ist für diese Arbeit hingegen zu detailliert. Aus diesem Grund stellen wir im Folgenden aus obigen Quellen drei Interpretationen für den Begriff Medium in Bezug zu Information zusammen, die auf die Bedürfnisse dieser Arbeit abgestimmt sind.

Interpretation 1 (Medium als dauerhafter Speicher von Information) *Die zentrale Fragestellung bei der Betrachtung eines Mediums als dauerhafter Speicher liegt auf dem Schwerpunkt, wo Information physikalisch gespeichert wird.*

[10] Z.B. das Medium Text für ein Repräsentationselement oder das Medium Papier für eine Präsentationselement.

Typische Beispiele für Medien im Sinne eines dauerhaften Speichers sind Papier, Festplatte und CD-ROM.

Interpretation 2 (Medium als Präsentationsschnittstelle von Information) *Bei der Auffassung eines Mediums als Präsentationsschnittstelle zwischen Information und Mensch steht die Frage im Vordergrund, über welche physikalische Realisierung und in welcher Richtung (nur Ausgabe von Information (Darstellung) oder auch Akzeptanz von Eingaben (Interaktion)) dies geschehen kann.*

Auf der Ausgabeseite stehen typischerweise Papier, Bildschirm und Lautsprecher zur Verfügung, auf der Eingabeseite Tastatur und Maus.

Interpretation 3 (Medium als innere/äußere Repräsentation von Information) *In dieser zweigeteilten Interpretation beschreibt der Begriff Medium zum einen die Art, in der Information physikalisch kodiert* [11] *ist (innere Repräsentation, Datenstruktur).*
Zum anderen beschreibt er die Art, in der Menschen Information mit ihren Sinnen[12] *aufnehmen (äußere Repräsentation),*

Nach dem zweiten Aspekt der letzten Interpretation (Medium als äußere Repräsentation von Information) können folgende Medien unterschieden werden:

Text	Daten in Form von Zeichen, Symbolen, Wörtern, Wortgruppen, Absätzen, Tabellen oder sonstigen Zeichenanordnungen, denen eine Bedeutung unterliegt und deren Interpretation durch den Leser im Wesentlichen auf seiner Kenntnis einer „natürlichen Sprache“ oder einer „künstlichen Sprache“ beruht [28].
Bild	Daten in Form von Pixelmatrizen.
Graphik	Daten in Form von Symbolen.
Bewegtbild	Zu einem Film zusammengesetzte Sequenz von Bildern.
Animation	Zu einem Film zusammengesetzte Sequenz von Graphiken.
Audio	Akustische Informationen, beispielsweise Töne, Sprache und Musik.
Video	Gemeinsame Anordnung von mehreren Bewegtbild-/Animations- und Audiosequenzen über einer gemeinsamen Zeitbasis (nach [19]).

Die in obiger Aufzählung genannten Medien lassen sich gemäß mehrerer Kriterien in voneinander unabhängige Gruppen aufteilen (siehe Abbildung 2.6):

[11] Beispielsweise können die einzelnen Buchstaben eines Textes in ASCII kodiert sein und Graphiken im gif- oder eps-Format vorliegen.

[12] Die für unsere weiteren Betrachtungen wichtigen Sinne sind das Sehen und das Hören.

- Bezüglich der *Zeit* kann man zwischen zeitunabhängigen (auch *diskret* genannten) Medien wie Text, Bild und Graphik auf der einen Seite, und den zeitabhängigen (auch *kontinuierlich* genannten) Medien wie Audio, Bewegtbild, Animation und Video auf der anderen Seite unterscheiden [50].

- Als weiteres Unterscheidungsmerkmal kann man die vorgestellten Medien bezüglich ihres *Ursprungs* klassifizieren. Die zwei möglichen Gruppen sind hierbei „natürlichen“ Ursprungs (*captured*, digitalisiert) und „künstlichen“ Ursprungs (*synthetisiert*, mit einem Rechensystem erstellt) [10]. Die Zuordnung der Medien zu diesen zwei Gruppen ist durch Abbildung 2.6 festgelegt. Es ist jedoch zu betonen, dass für die Instanzen natürlicher Medien der Übergang hin zu einer Instanz eines synthetisierten Mediums bei der Bearbeitung mit Rechensystemen fließend ist.

- Ein letztes nur für das Medium Video zutreffendes Klassifizierungskriterium ist die Eigenschaft, aus mehreren anderen synchronisierten Medien *zusammengesetzt* zu sein [19].

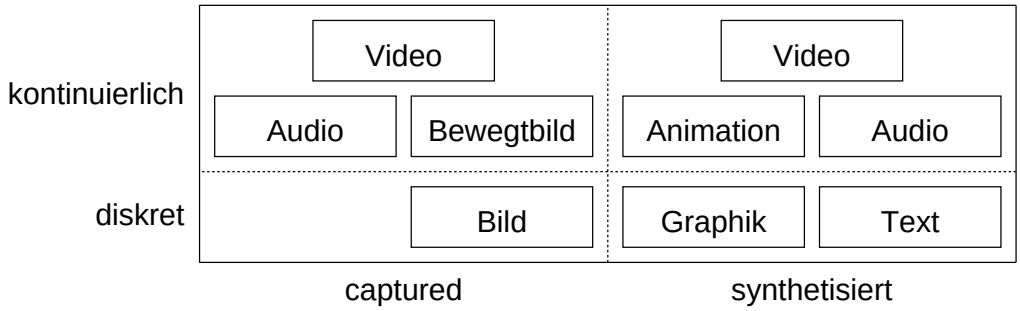

Abbildung 2.6: Klassifikation von Medien

Anknüpfend an Definition 1 zu Dokumenten ergibt sich somit, dass Daten und Struktur (logische Objekte) eines Dokumentes, die in einem Speichermedium in Form von inneren Repräsentationen abgelegt sind, durch ein Präsentationsmedium über Layoutelemente (äußere Repräsentationen) den Nutzern zugänglich gemacht werden (siehe Abbildung 2.7).

Multimedia

An dieser Stelle ist es unumgänglich, einen kurzen Überblick über die kombinierte Verwendung mehrerer Medien in einem Dokument in Zusammenhang mit dem Trend-Begriff *Multi*media[13] [Multi- (lat.:viel), als Präfix] zu geben. Bei [25] findet sich dazu folgende Aussage:

> „Multimedia systems coherently handle diverse media types, including text, graphics, electronic ink, animation, images, video, and audio. ... However, there is no consensus on what multimedia systems really are: like object-oriented programming, everyone talks about it but no one knows what it is.“

[13] Im deutschen Sprachraum wurde der Begriff „Multimedia“ im Jahre 1995 zum Wort des Jahres gewählt. [20]

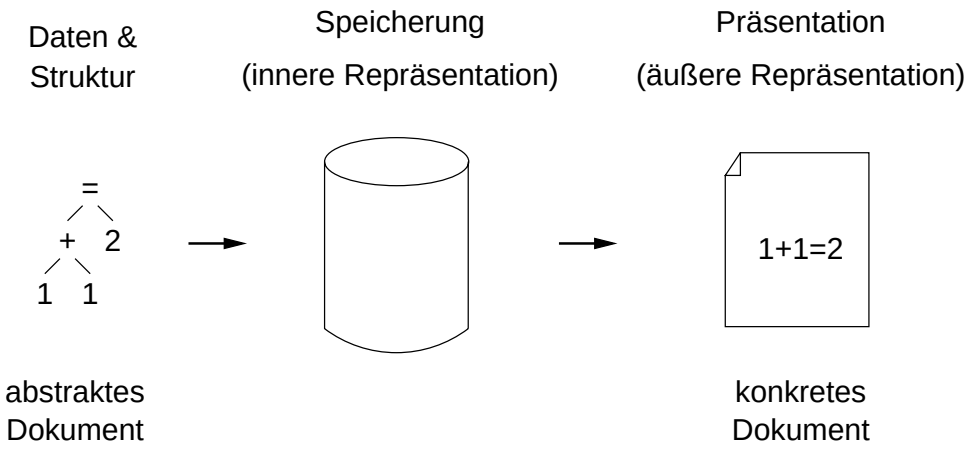

Abbildung 2.7: Vom abstrakten zum konkreten Dokument

In Einklang mit dieser Aussage preisen sich viele Produkte mit der Behauptung an, sie seien multimedial, indem sie sich auf folgendes abstützen: Als multimedial könnte man auf Basis der Wortbestandteile von Multimedia bereits solche Dokumente bezeichnen, die mehr als ein Medium verwenden. Somit wäre bereits ein Dokument, das lediglich Text und Graphik beinhaltet, ein Multimedia-Dokument.

Ein derartiges, streng quantitatives Vorgehen, das den Titel Multimedia ausschließlich in Hinblick auf die Anzahl der verwendeten Medien vergibt, wird jedoch der Bedeutung des Begriffes Multimedia, wie er heute umgangssprachlich gebraucht wird, nicht gerecht. Somit gilt es, eine qualitative Festlegung für den Begriff Multimedia zu finden. In der Literatur finden sich hierzu drei Kriterien, die im Folgenden zusammengestellt werden:

Kriterium 1 (Kombination von Medien) *Wie bereits angeführt, rechtfertigen* triviale[14] *Kombinationen mehrerer Medien noch nicht die Verwendung des Begriffes Multimedia. Deshalb erheben wir die Forderung, dass in multimedialen Dokumenten zugleich diskrete als auch kontinuierliche Medien verwendet werden müssen.*

Kriterium 2 (Unabhängigkeit der Medien) *Die Forderung nach Unabhängigkeit der Medien läßt sich am einfachsten am Beispiel Video verdeutlichen: Die beiden hier vereinigten Medien Audio und Bewegtbild/Animation sind über eine gemeinsame Zeitbasis starr aneinander gekoppelt und erscheinen uns als eine untrennbare Einheit. Eine Eigenschaft, die sich mit dem Begriff Multimedia nicht in Einklang bringen läßt – einen Kino oder Fernsehfilm fassen wir nicht als multimedial auf.*

Kriterium 3 (Interaktivität/Rechnergestützte Präsentation) *Im Gegensatz zu den ersten beiden Kriterien baut dieses letzte Kriterium für den Begriff Multimedia nicht auf dessen Wortbestandteilen selbst auf. Durch die im vorherigen Punkt aufgestellte Forderung, Medien unabhängig und frei voneinander in Präsentationen kombinieren und nutzen zu können, stellt sich*

[14] Z.B. die ausschließlich gleichzeitige Verwendung von nur Text und Graphik in einem Dokument.

die Frage, wie man diese neu gewonnene Freiheit nutzen kann. Die Antwort darauf ist die interaktive Steuerungsmöglichkeit von Multimedia-Produkten durch die Nutzer, die durch eine rechnergestützte Präsentation erst möglich wird.

Definition 14 (Multimedia) *Das Attribut Multimedia vergeben wir nur dann für ein Objekt*[15]*, wenn die drei Multimedia-Kriterien* Kombination von Medien, Unabhängigkeit der Medien *und* Interaktivität/Rechnergestützte Präsentation *für dieses Objekt erfüllt sind.*

2.1.4 Ausprägungen von Dokumenten

Mit Einführung der verschiedenen Strukturarten für Dokumente und der Diskussion des Begriffes Medium öffnet sich der Weg, Dokumente bezüglich dieser Begriffe zu kategorisieren.

Einstufung aufgrund der verwendeten Medien

Die einfachste Dokumentart, die sich aufgrund der unterschiedlichen Verwendungsmöglichkeiten von Medien ergibt, sind Text-Dokumente. Sie enthalten als Repräsentationsmedium nur Text und sind entweder auf Papier gedruckt oder am Bildschirm dargestellt[16] und nicht interaktiv[17] nutzbar. In dieselbe Klasse können auch Dokumente eingeordnet werden, die zusätzlich zum Repräsentationsmedium Text die Repräsentationsmedien Bild und Grafik verwenden; alle derart ausgezeichneten Dokumente beinhalten nur diskrete Medien zur Repräsentation. Gemäß der Definition des Begriffes Multimedia ist es nicht angezeigt, diese Dokumente trotz der Verwendung mehrerer Medien als *multimediale Dokumente* zu bezeichnen. Auch die Bezeichnung *monomediale Dokumente* ist für diese Klasse von Dokumenten nicht zutreffend. Mangels eines treffenden existierenden Begriffes in der Literatur wollen wir diese Art von Dokument in Anlehnung an ODA mit *Büro*[18] *Dokument* bezeichnen:

Definition 15 (Büro Dokument) *Ein Büro Dokument enthält als Repräsentationselemente nur Text, Bild und Graphik Layoutelemente, die durch diskrete Medien realisiert werden. Es wird in der Regel entweder auf Papier oder einem Bildschirm präsentiert.*

In folgender Definition legen wir fest, unter welchen zusätzlichen Bedingungen ein Büro Dokument als Multimedia-Dokument bezeichnet werden kann. Wir greifen dabei auf die zur Kennzeichnung des Begriffes Multimedia eingeführten Kriterien zurück.

[15] Z.B. Dokumente und Applikationen.

[16] Selbstverständlich sind noch weitere Präsentationselemente möglich. Sie werden hier jedoch nicht explizit aufgeführt.

[17] Außer z.B. der Tatsache, dass durch ihren Inhalt bei einer Bildschirmpräsentation gescrollt werden kann.

[18] Die meisten in Büros auftretenden Dokumente beinhalten nur die Repräsentationsmedien Text, Bild und Graphik.

Definition 16 (Multimedia-Dokument) *Ein Multimedia-Dokument ist statisch durch die Integration von mindestens einem diskreten und einem kontinuierlichem Medium, die voneinander unabhängig sind, gekennzeichnet.*
Hinzu kommt, dass die beteiligten Medien rechnergestützt interaktiv unabhängig voneinander durch die Nutzer in einer geeigneten Applikation (Multimedia-System) gesteuert werden können.

Mit obiger Definition ist offensichtlich, dass Multimedia-Dokumente nicht losgelöst von den Applikationen betrachtet werden können, mit deren Hilfe sie von den Nutzern betrachtet werden. Bedingt durch die rechnergestützte interaktive Nutzung kann ein Dokument erst dann zu einem Multimedia-Dokument werden, wenn es in einem Multimedia-System verwendet wird.

Die Definitionen für Büro und Multimedia Dokumente stützen sich nur auf die Verwendung von Medien und die interaktive Steuerung von den in Dokumenten beinhalteten Medien ab. In den folgenden Definitionen spielt auch die Struktur der Dokumente für deren Klassifizierung eine Rolle.

Einstufung aufgrund der verwendeten Medien und der Struktur

Bezüglich ihrer interaktiven Nutzung stehen die sogenannten *Hypertext-* und *Hypermedia-Dokumente* eng in Verbindung mit Multimedia-Dokumenten. Alle genannten Begriffe bezeichnen Dokumente, die interaktiv gesteuert durch die Nutzer betrachtet werden können und somit Systeme zu deren Betrachtung benötigen. Offen ist jedoch, worauf sich die interaktive Nutzung der Hypertext- und Hypermedia-Dokumente stützt.

Nach [50] weisen Hypertext- und Hypermedia-Dokumente als eine wesentliche Eigenschaft eine stark ausgeprägte *nichtlineare Informationsverkettung*, die wir auch mit Verweisstruktur bezeichnet haben, auf. Diese Verweisstruktur der Dokumente wird in Hypertext- und Hypermedia-Systemen benutzt, um den Nutzern einen interaktiv beeinflussten, nichtlinearen[19] Nutzungspfad[20] durch die Dokumente zu ermöglichen. Realisiert wird dies üblicherweise, indem aufgrund der Verweisstruktur in der Bildschirmpräsentation von Hypertext- und Hypermedia-Dokumenten sensitive *Ankerregionen*[21] eingeführt werden, anhand derer die Nutzer nichtlinear durch das gegebene Dokument zu Zielpositionen navigieren können.

Definition 17 (Hypertext-Dokument) *Ein Hypertext-Dokument ist ein durch das Vorhandensein einer stark ausgeprägten Verweisstruktur ausgezeichnetes Dokument, das in einer geeigneten Applikation (Hypertext-System) den Nutzern zur nichtlinearen interaktiven Nutzung zur Verfügung gestellt wird.*

[19] Nicht in der sequentiellen Reihenfolge der linearisierten Struktur, die bei einem normalen auf Papier gedruckten Dokument von oben nach unten und von links nach rechts geht, sondern der Verweisstruktur folgend.
[20] Folge der Layoutelemente, die ein Nutzer eines Dokumentes konsumiert.
[21] Z.B. von den Nutzern im Dokument mit der Maus anklickbare Bereiche.

Irreführend bei dem Begriff Hypertext in seiner Verwendung gemäß neuerer Literatur ist sicherlich das Suffix „-text“, legt es doch nahe, in einem Hypertext sei nur das Medium Text[22] erlaubt. Tatsache ist jedoch, dass nach heutiger Auffassung in einem Hypertext-Dokument beliebige Medien verwendet werden dürfen. Einschränkendes Kriterium ist ausschließlich das Vorhandensein einer ausgeprägten Verweisstruktur und deren Verwendung zur Navigation durch das Dokument mit Hilfe eines interaktiven Systems. Im Sinne einer konsequenten Begriffsbildung wäre es deshalb naheliegend, Hypertext-Dokumente schlicht als *Hyper-Dokumente* zu bezeichnen.

Offen ist an dieser Stelle nun noch die Definition von Hypermedia-Dokumenten:

Definition 18 (Hypermedia-Dokument) *Ein Hypermedia-Dokument ist ein durch das Vorhandensein einer stark ausgeprägten Verweisstruktur ausgezeichnetes Multimedia-Dokument, das in einer geeigneten Applikation (Hypermedia-System) den Nutzern zur nichtlinearen interaktiven Nutzung zur Verfügung gestellt wird.*

Die letzten beiden Definition zusammengenommen machen offensichtlich, dass jedes Hypermedia-Dokument auch ein Hypertext-Dokument ist. Weiterhin ist anzumerken, dass nach obigen Definitionen ein Hypertext- bzw. Hypermedia-Dokument als solches nur dann bezeichnet werden kann, wenn es in einer Präsentation mit Hilfe eines Hypertext- bzw. Hypermedia-Systems interaktiv genutzt wird.

Im Sinne einer konsequenten Begriffsbildung würde man analog zu dem Vorschlag bei Hypertext-Dokumenten den Begriff Hypermedia-Dokument in *Hyper-Multimedia-Dokument* umbenennen. Die Vorsilbe „Hyper“ würde dann in beiden Fällen anzeigen, dass ein Dokument in einem System vorliegt, durch das die Nutzer mit Hilfe einer gegebenen Verweisstruktur interaktiv nichtlinear navigieren können. In der Literatur werden derartige Hyper-Dokumente auch als *nichtlineare Dokumente* bezeichnet:

Definition 19 (Nichtlineares Dokument) *Ein Dokument, dessen Verweisstruktur in geeigneten Systemen zur nichtlinearen interaktiven Navigation verwendet wird, bezeichnen wir als nichtlineares Dokument.*

Es stellt sich nun die Frage, wie die nichtlineare Verweisstruktur von gegebenen Dokumenten in nicht mit Hilfe eines Systems interaktiv nutzbaren Dokument-Präsentationen verwertet werden kann. Tatsache ist, dass jede in einem Dokument mit einer hierarchischen Struktur versehene Information in eine linearisierte Struktur serialisiert werden muss, bevor sie beispielsweise in eine auf Papier gedruckte, statische Präsentation umgesetzt werden kann. Üblicherweise wird hierzu ein depth-first Durchlauf durch die gegebene hierarchische Struktur des Dokumentes gewählt. Problematisch gestaltet sich jedoch die Umsetzung einer gegebenen nichtlinearen Verweisstruktur, die sich einem derartigen Durchlauf entzieht.

Die Lösung für gedruckte Dokumente bietet sich den Nutzern meist in Form von Querverweisen, Inhaltsverzeichnissen, Indexen und Fußnoten dar. Die Verweisstruktur wird also

[22] Eine Auffassung, die sich in frühen Publikationen zu diesem Thema auch noch findet [44].

in Verbindung mit den weiteren im Dokument vorhandenen Daten und Strukturinformationen genutzt, um diese generierten Repräsentationen in konkreten Dokumenten zu erzeugen. Ihre Nutzung bildet die interaktiv nichtlineare Nutzung der Verweisstruktur von Hyper-Dokumenten in geeigneten Präsentationssystemen nach.

Definition 20 (Lineares Dokument) *Ein lineares Dokument ist ein Dokument, dessen evtl. gegebene Verweisstruktur nicht mit Hilfe einer Applikation von den Nutzern zur nichtlinearen interaktiven Navigation durch das Dokument genutzt werden kann.*

Hervorzuheben ist, dass die letzte Definition nicht die interaktive Nutzung eines Dokumentes in einer Bildschirmpräsentation ausschließt, in der beispielsweise mit Hilfe einer Scrollmöglichkeit der sichtbare Dokumentausschnitt bestimmt werden kann. Lediglich die programmgestützte nichtlineare interaktive Navigationsmöglichkeit ist ausgeschlossen. Weiterhin ist anzumerken, dass ein nichtlineares Lesen eines gedruckten linearen Dokumentes, indem z.B. vom Fließtext ausgehend eingestreute Fußnoten gelesen werden oder mit Hilfe des Inhaltsverzeichnisses gezielt eine Seite aufgeschlagen wird, es uns nicht gestattet, dieses Dokument als nichtlineares Dokument zu kennzeichnen.

2.2 Digitale Dokumenterstellung

Sowohl der Begriff Textverarbeitung als auch der mit ihm synonym verwendete Begriff „word processing“ legen mit ihren Wortbestandteilen nahe, dass die von ihnen bezeichnete Aufgabenstellung die rechnergestützte Verarbeitung von Text-Dokumenten ist. Tatsächlich wird mit ihnen jedoch die rechnergestützte Verarbeitung von Dokumenten bezeichnet, die mehr als nur das Medium Text beinhalten. Waren diese Ausdrücke zu Beginn der rechnergestützten Verarbeitung von Dokumenten, in der tatsächlich nur Text verarbeitet wurde, korrekt, so werden sie heute nurmehr aus traditionellen Gründen verwendet. Beachtet man aber den gewichtigen Aspekt eines Dokumentes, Informationsträger zu sein, liegt es nahe, auch die traditionellen Gründe nicht mehr als Argument für die Verwendung des inhaltlich mißverständlichen Begriffes Textverarbeitung zählen zu lassen. Um diesen Aspekt der Information zu betonen, sollte man den Begriff Dokument in den Vordergrund stellen. Der Vorgang der Erstellung eines Dokumentes würde somit mit dem Begriff *Dokumenterstellung* bezeichnet werden.

Definition 21 (Dokumenterstellung) *Unter Dokumenterstellung versteht man den Vorgang, der alle Tätigkeiten beinhaltet, die zur Erstellung eines Dokumentes notwendig sind.*

Im allgemeinen Fall umfaßt die Dokumenterstellung die frühe Phase der Ideenfindung, geht über die Recherche und die eigentliche (dauerhaft gespeicherte) Erfassung und Aufbereitung des Dokumentes und endet mit der Veröffentlichung desselbigen.

Bevor wir inhaltlich auf die digitale Dokumenterstellung eingehen, ist erst einmal zu klären, welchen Arbeitsbereich wir überhaupt mit diesem Begriff in dieser Arbeit abstecken wollen. In

der Definition zur Dokumenterstellung haben wir festgelegt, dass wir bei der Erstellung von Dokumenten alle Tätigkeiten von der Informationssuche bis hin zur Publikation mit einschließen. Die in dieser Arbeit entwickelten Methoden befassen sich jedoch nicht mit den frühen und späten Phasen dieser Definition (Informationssammlung und Vertrieb). Folglich verstehen wir in dieser Arbeit unter digitaler Dokumenterstellung nur die Unterstützung der Arbeitsabläufe von der Eingabe aller für ein Dokument benötigten Inhalte (Daten, Struktur, Repräsentation und Präsentation) bis hin zur Aufbereitung dieser Inhalte für eine den Nutzern zugängliche Präsentation:

Definition 22 (Digitale Dokumenterstellung) *Unter digitaler Dokumenterstellung verstehen wir in dieser Arbeit den durch ein digitales Rechensystem unterstützten Vorgang, der alle Tätigkeiten von der Eingabe aller für ein Dokument benötigten Inhalte bis hin zu deren Aufbereitung für eine den Nutzern zugängliche Präsentation, beinhaltet.*

Wie wir in 2.1.1 gesehen haben, bestehen Dokumente aus den vier Komponenten Daten, Struktur, Repräsentation und Präsentation. Als Antwort auf die Frage, welche Informationen zur rechnergestützten Erstellung von Dokumenten erfaßt werden müssen, stellen wir im Folgenden die zwei konkurrierenden Paradigmen zur digitalen Erstellung von Dokumenten vor, die momentan in verfügbaren Systemen eingesetzt werden. Ihre einzige Gemeinsamkeit ist die selbstverständliche Notwendigkeit der Datenerfassung. Wie wir sehen werden, beschäftigt sich der erste der beiden Ansätze dann zusätzlich zur reinen Datenerfassung ausschließlich mit der Erfassung von Repräsentations- und Präsentationsinformation, während sich der zweite Ansatz rein auf die Erfassung von Strukturinformation stützt.

2.2.1 Graphisches Auszeichnen

Der Ansatz des graphischen Auszeichnens[23] ist uns von der Benutzung konventioneller Dokumentbearbeitungssysteme wie z.B. Microsoft Word, Quark Xpress, Adobe FrameMaker und TeX her bekannt. Typischerweise gibt man bei Verwendung dieser Systeme zur Erstellung eines Dokumentes zunächst den gewünschten Text ein (Dateneingabe). Im Anschluss daran oder auch zeitlich verschränkt hierzu zeichnet man gewisse Textbereiche (z.B. durch Markieren mit der Maus oder in textueller Form durch setzen von Markierungen) aus und weist ihnen dann für bestimmte graphische Attribute[24] Werte zu (graphisches Auszeichnen).

Das hierbei verwendete Modell betrachtet ein Dokument als syntaktisch unterscheidbare Sequenz von Daten und sogenannten *Tags*. Tags treten immer paarweise auf und markieren die zwischen ihnen liegenden Bereiche im Dokument. Sie sind in der Regel systemspezifisch und werden zum graphischen Gestalten des Dokumentes verwendet. Sie geben den Autoren unter anderem die Möglichkeit Textstücke kursiv oder fett sowie in verschiedenen Schriftarten

[23] Im englischen Sprachraum auch „Procedural Markup“ genannt, da das Layout der Dokumente über im Text eingefügte Marken bestimmt wird, die angeben, welche Prozeduren (beispielsweise in einem Drucker) zur Umsetzung des Layouts verwendet werden [33].

[24] Z.B. Font, Farbe, Einrückung.

und Farben zu setzen, Absatzeinzüge zu spezifizieren und Randbreiten zu bestimmen (siehe Abbildung 2.8). Die zur graphischen Gestaltung eines Dokumentes verwendeten Tags werden in ihrer Gesamtheit auch *Markup* genannt.

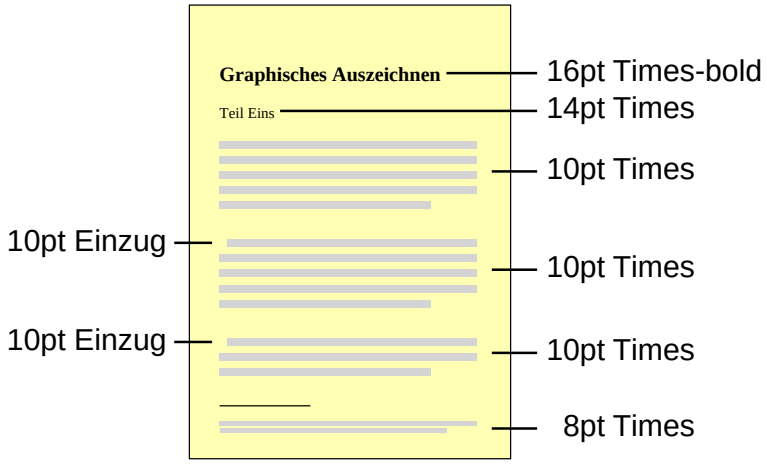

Abbildung 2.8: Graphisches Auszeichnen von Dokumenten

Zusammenfassend kann man festhalten, dass das Ziel des graphischen Auszeichnens von Dokumenten die kombinierte Erstellung von Inhalten und deren fest zugewiesener Repräsentationen sowie Präsentation ist. Die Strukturinformation des Dokumentes wird bei diesem Ansatz nur visuell kodiert, sie ist maschinell nicht direkt zugänglich, kann in der Regel aber vom Menschen dekodiert werden.

2.2.2 Logisches Auszeichnen

Ziel des logischen Auszeichnens[25] von Dokumenten ist, die Strukturierung (hierarchische- und Verweisstruktur) eines Dokumentes möglichst vollständig explizit zu erfassen (siehe Abbildung 2.9). Wie bei der graphischen Auszeichnung von Dokumenten erfaßt man beim logischen Auszeichnen zunächst die Daten des Dokumentes und zeichnet dann mit Hilfe von Tags Bereiche aus.

Ähnlich zu der Betrachtungsweise bei der graphischen Auszeichnung von Dokumenten betrachtet man bei der logischen Auszeichnung ein Dokument als syntaktisch unterscheidbare Sequenz von Daten und Tags. Im Vordergrund steht aber diesmal das Ziel, mit Hilfe der paarweise verwendeten Tags auszuzeichnen, welche Aufgabe eine gewisse Information in einem Dokument übernimmt und nicht, wie sie repräsentiert wird. Typischerweise definiert man also inhaltlich, welche Dokumentbereiche z.B. Überschriften, Texte, Fußnoten, u.s.w., sind. Mit Hilfe des logischen Auszeichnens werden in einem Dokument dessen logische Objekte und somit dessen logische Struktur festgelegt.

Bekannt ist uns dieser Ansatz aus Systemen wie z.B. LaTeX, Adobe FrameMaker und Microsoft Word. Die beiden letztgenannten Systeme bieten, wie bereits vorgestellt, zusätzlich

[25] Im englischen Sprachraum auch „Descriptive Markup“ [23] und „Declarative Markup“ [33] genannt.

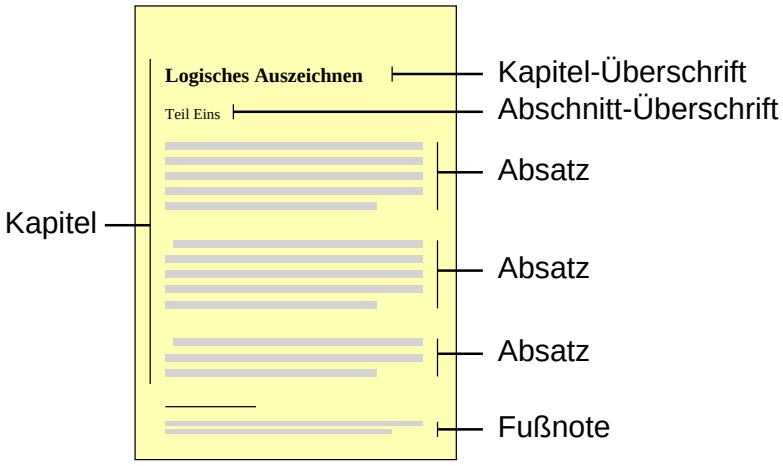

Abbildung 2.9: Logisches Auszeichnen von Dokumenten

zum hier vorgestellten Ansatz des logischen Auszeichnens den Ansatz des graphischen Auszeichnens an. Wesentlicher Grund für diese Doppelgleisigkeit bei den genannten Systemen ist der Marktdruck, der von den Käufern dieser Systeme ausgeht.

2.2.3 Graphisches vs. Logisches Auszeichnen

Nach diesem Überblick über die zwei unterschiedlichen Verfahren zur Auszeichnung von Dokumenten lohnt es sich, nochmals Abbildung 2.1 zu betrachten und schematisch zu untersuchen, welche Information mit Hilfe der zwei unterschiedlichen Vorgehensweisen erfasst werden.

Wie in Abbildung 2.10 zu erkennen ist, fehlt beim Vorgehen der graphischen Auszeichnung jegliche Information bezüglich der Struktur eines Dokumentes, wohingegen bei der Methode des logischen Auszeichnens jegliche Information bezüglich der Repräsentation und Präsentation eines Dokumentes vernachlässigt wird.

Eine erste Feststellung, die man aus dieser Analyse ziehen kann, ist die Erkenntnis, dass graphisch ausgezeichnete Dokumente aufgrund der in ihnen explizit enhaltenen Layoutinformation unmittelbar mit den dafür vorgesehenen Werkzeugen publiziert werden können. Bei logisch ausgezeichneten Dokumenten ist dies nur in Systemen möglich, denen die Semantik der in den Dokumenten zur Auszeichnung verwendeten Tags bekannt ist. Für allgemeine logisch ausgezeichnete Dokumente ist die Publikationsfähigkeit in einem derartigen System nicht gegeben.

Graphisches Auszeichnen

Untersucht man zunächst die graphische Auszeichnungsmethodik weiter, stellt man aber sofort fest, dass mit ihrer Hilfe nur exakt eine Repräsentation für ein Dokument festgelegt werden kann – weitere sind nur durch ein Kopieren des Dokumentes und den damit verbundenen Nachteilen

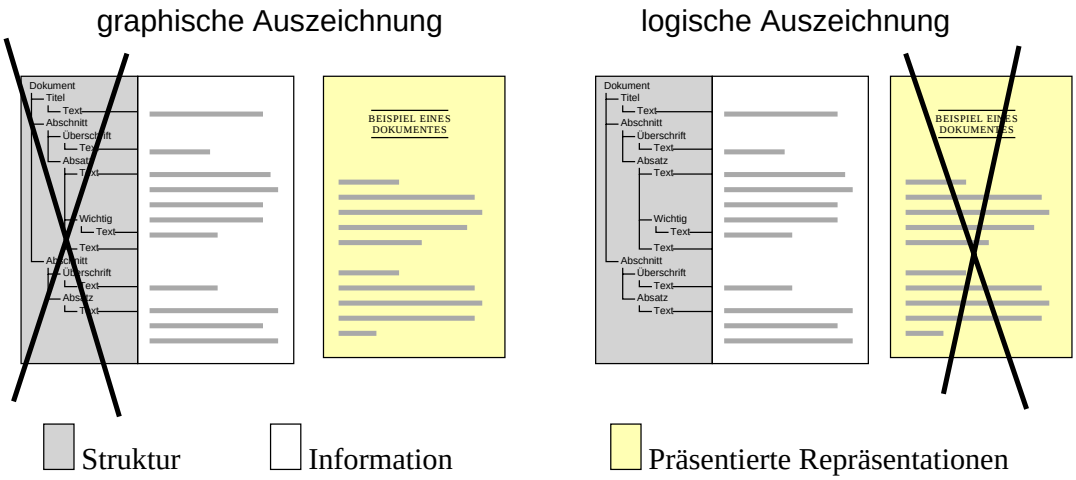

Abbildung 2.10: Vergleich graphische vs. logische Auszeichnung

(zwei getrennt zu pflegende Dokumente mit dem selben Inhalt, der konsistent gehalten werden muss), möglich.

Ein weiterer nachteiliger Punkt der graphischen Auszeichnungsmethodik liegt in der Verantwortung der Dokumentersteller begründet, wenn sie Dokumente konsistent zu vorgegebenen Richtlinien erstellten müssen. Es liegt bei diesem Vorgehen vollkommen in ihrer Verantwortung, sich konsequent an die gegebenen Richtlinien zu halten. Hinzu kommt, dass bei einer Änderung der vorgegebenen Richtlinien eine massive Umformatierung bestehender Dokumente nötig werden kann, deren konsistente Durchführung technisch und finanziell sehr aufwendig sein kann. Analog kann der Umstieg von einem System auf ein anderes einen extrem hohen Aufwand nach sich ziehen.

Auch stellt sich die Frage, wie z.B. Inhaltsverzeichnisse und Indexe mit Hilfe des graphischen Auszeichnens erstellt werden können. Als einzige Lösung bietet es sich beim reinen graphischen Auszeichnen an, derartige Dokumentbestandteile von Hand zu erstellen und dann graphisch auszuzeichnen. Es ergeben sich somit redundante Dokumentbestandteile, die konsistent gehalten werden müssen.

Zuletzt bleibt die Zeit zu untersuchen, die zur Erstellung eines graphisch ausgezeichneten Dokumentes verwendet wird. An dieser Stelle kann selbstverständlich nur ein grober Bereich genannt werden, da je nach Art des erstellten Dokumentes unterschiedlich viel Aufwand für die Aufbereitung der Repräsentation eines Dokumentes verwendet werden wird. Nach [4] bewegt sich dieser Bereich jedoch zwischen 15% bis 50% der gesamten zur Erstellung eines Dokumentes benötigten Zeit – und ist damit unverhältnismäßig hoch.

Logisches Auszeichnen

Wechselt man nun zu der Untersuchung der logisch ausgezeichneten Dokumente, gilt es zunächst das Problem zu lösen, wie derart erstellte Dokumente überhaupt verarbeitet werden können. Üblicherweise wird dieses Problem mit Hilfe sogenannter *Stylesheets* gelöst, die streng

getrennt von dem eigentlichen Dokument verwaltet werden können. Sie legen fest, wie einzelne logisch markierte Dokumentbestandteile in einem konkreten Dokument präsentiert werden sollen. Durch diese Trennung von Dokumentinhalt und dessen Layout wird es im Gegensatz zum Ansatz der graphischen Auszeichnung zumindest prinzipiell möglich, ein gegebenes logisch ausgezeichnetes Dokument beliebig oft auf unterschiedliche Weise zu publizieren, ohne in das Dilemma des Kopierens zu verfallen.

Auch weitere Nachteile, die bei graphisch ausgezeichneten Dokumenten aufgefallen sind, fallen bei der logischen Auszeichnung weg. Kurz im Überblick genannt sind dies die konsistente Gestaltung eines Dokumentes sowie die Neugestaltung im Falle einer Änderung der vorgegebenen Richtlinien. Schwierig ist es allerdings an dieser Stelle eine Aussage über den Aufwand beim Umstieg auf ein anderes System zu machen, solange man keine Aussage über das Format (standardisiert oder nicht) der logisch ausgezeichneten Dokumente trifft. Setzt man voraus, dass die logisch ausgezeichneten Dokumente nicht konvertiert werden müssen bzw. automatisch konvertiert werden können (aufgrund ihrer reichen Strukturinformation ist dies naheliegend), so ist bei einem Umstieg lediglich ein von der Anzahl der vorhandenen Dokumente unabhängiger und damit konstanter Aufwand (Konvertierung des Stylesheets) zu leisten. Ein im Vergleich zu graphisch ausgezeichneten Dokumenten erheblicher Vorteil.

Abschliessend ist darauf hinzuweisen, dass aus der logischen Auszeichnung von Dokumenten mit Hilfe von Stylesheets ein konkretes Dokument erzeugt werden kann, wohingegen aus einer Repräsentation die zugehörige logische Auszeichnung nicht immer gewonnen werden kann. Begründet liegt diese Feststellung in der nicht immer eindeutigen Repräsentation für verschiedene logische Bereiche in einem Dokument. Beispielhaft kann hier die Verwendung von Kursivschrift herangezogen werden: In einem Ausgangsdokument seien alle Hervorhebungen und Definitionen kursiv gedruckt. Soll die Repräsentation für Hervorhebungen auf Fettschrift geändert werden, fehlt für ein automatisches Vorgehen die Unterscheidungsmöglichkeit der Hervorhebungen von den Definitionen.

2.2.4 Existierende Systeme

Nachdem wir nun die beiden wesentlichen Paradigmen zur digitalen Erfassung von Dokumenten vorgestellt haben, gilt es deren Verwendung in aktuell verfügbaren Systemen zu untersuchen.

Als erstes Ergebnis einer derartigen Analyse ist festzuhalten, dass die meisten am Markt verfügbaren Systeme eine Mischung aus den Ansätzen graphisches und logisches Auszeichnen darstellen. Systeme, die in Reinform die graphische Auszeichnung implementieren, finden sich nicht mehr, wohl aber einige wenige Systeme, die konsequent die logische Auszeichnung unterstützen.

Schwerpunktmäßig auf graphischer Auszeichnung basierende Systeme

Wie wir bereits bei der Untersuchung des graphischen Auszeichnens festgestellt haben, kommt es bei diesem Vorgehen bereits bei der Erstellung von Inhaltsverzeichnissen zu Schwierigkei-

ten. Der Grund hierfür liegt allgemein in der Tatsache, dass ein beliebiges gegebenes Software-System nicht in der Lage ist, aufgrund eines gegebenen ausschließlich graphisch ausgezeichneten Dokumentes in eindeutiger Weise die Dokumentteile zu bestimmen, die in das Inhaltsverzeichnis aufgenommen werden sollen. Aufgrund der selben Tatsache können auch weitere, heute in Dokumentbearbeitungssystemen als selbstverständlich betrachtete Aufgaben, in Systemen mit rein graphischer Vorgehensweise nicht gelöst werden.

Aus diesem Grund wurde in alle Systeme, die schwerpunktmäßig auf dem Paradigma der graphischen Auszeichnung basieren, die Idee des logischen Auszeichnens in mehr oder weniger konsequenter Weise integriert. Das typische Beispiel ist die Implementierung der Möglichkeit, eine logische Auszeichnung auf der Ebene von Absätzen durchführen zu können. Durch sie wird es den Dokumenterstellern ermöglicht, beispielsweise Absätze als Überschriften in verschiedenen Ebenen zu klassifizieren. Eine Eigenschaft die es Systemen ermöglicht, automatisch z.B. Inhaltsverzeichnisse zu erstellen. In neueren Versionen ist es sogar möglich, Textbereiche unterhalb der Absatzebene logisch auszuzeichnen und somit festzulegen, ob ein gewisser Textteil etwas „wichtiges“ oder ein „Produktname“ ist.

Eine Auszeichnung oberhalb der Absatzebene ist allerdings nicht möglich. Folglich kann mit einem derartigen System keine Struktur eines Dokumentes modelliert werden, das aus mehreren Gedichten besteht, die wiederum aus mehreren Versen bestehen. Somit ist die erzielbare hierarchische Struktur von ihrer Tiefe her begrenzt, eine Tatsache, die vielen Dokumenten aber nicht entspricht [7].

Mit Einführung der logischen Auszeichnung in graphisch basierte Systeme wird es zwar einerseits notwendig, auch hier Stylesheets zur Festlegung der Behandlung der logisch ausgezeichneten Dokumentteile zu verwenden. Andererseits gewinnt man (aufgrund der nur partiellen Unterstützung jedoch nur in gewissen Grenzen) wie bereits bei den Vorteilen des logischen Auszeichnens vorgestellt wurde, an Flexibilität bei der einheitlichen Gestaltung und nachträglichen Umgestaltung von Dokumenten.

Die auf einer Kombination der beiden vorgestellten Auszeichnungsparadigmen basierenden Systeme sind meist mit einer Benutzungsoberfläche ausgestattet, die dem WYSIWYG[26] Konzept entspricht. Die bearbeiteten Dokumente können deshalb meist exakt in der Repräsentation gedruckt werden, in der sie am Bildschirm dargestellt werden. Auch können die bearbeiteten Dokumente direkt in der WYSIWYG-Präsentation, in der sie am Bildschirm angezeigt werden, editiert werden (indirekte Eingabe). In vielen Implementierungen gängiger Systeme bewirkt diese Tatsache aber offensichtlich eine Abstützung der Systeme auf echtzeitfähige und interaktions-geeignete Repräsentationalgorithmen, die die Darstellungsqualität des konkreten Dokumentes (aufgrund des WYSIWYG-Paradigmas auch in der gedruckten Version) sehr niedrig halten.

Repräsentanten für Systeme aus diesem Bereich belegen sich typischerweise selbst mit der Aussage, ein Word-Processing System zu sein, oder kommen aus dem Bereich des Desktop Publishing.

[26] *W*hat *Y*ou *S*ee *I*s *W*hat *Y*ou *G*et.

Auf logischer Auszeichnung basierende Systeme

Die wenigen verfügbaren Editoren, die rein mit der logischen Auszeichnung von Dokumenten arbeiten, basieren im Gegensatz zu den auf graphischer Auszeichnung beruhenden Systemen in der Regel nicht auf dem WYSIWYG Prinzip. Statt dessen wird der Inhalt der bearbeiteten Dokumente in einer „Pseudo-WYSIWYG“ Darstellung angezeigt und bearbeitet, die zwar ähnlich zu einer WYSIWYG Darstellung ist, nicht jedoch deren Perfektion erreichen will. Man verspricht sich von diesem Vorgehen eine Vereinfachung des Erstellungsprozesses an sich, da sich die Dokumentersteller besser auf ihre eigentliche Arbeit konzentrieren können: Die Formulierung der Dokumentdaten und deren Struktur. Coombs u.a. [18] schreiben hierzu: „One of the more subtle advantages of descriptive markup is it supports authors in focusing on the structure and content of documents.“. Riehm [48] und andere kommentieren diese Festellung jedoch mit der Aussage, dass ihnen keine empirische Untersuchung bekannt ist, wonach diese Art der Dokumenterstellung einen positiven Effekt hat.

Im einfachsten Fall findet die Erstellung logisch ausgezeichneter Dokumente mit Hilfe von konventionellen Texteditoren statt, wobei das Dokument im Prinzip als eine Sequenz von Text und klammerartig eingestreuten Tags eingegeben wird. Da die Tags in diesem Fall vom Editor nicht vom normalen Text unterschieden werden können, werden sie wie gewöhnlicher Text angezeigt und sind auch nicht systemgestützt editierbar (*pseudoverbale Eingabe* [8]).

Im benutzerfreundlicheren Fall kann die Eingabe aber (mit Unterstützung von Stylesheets) auch in einer Pseudo-WYSIWYG Darstellung erfolgen, bei der der Dokumenttext ähnlich zu der endgültigen Fassung angezeigt wird (z.B. Überschriften in größerer Schrift als nomaler Text), die Tags (die vom System erkannt werden können) hingegen über Symboldarstellungen (*symbolische Eingabe* [8]). In derartigen Systemen sind die Tags typischerweise systemunterstützt editierbar.

Die rein auf der logischen Auszeichnung beruhenden Editoren ermöglichen die Spezifizierung wesentlich „reicherer“ hierarchischer Strukturen, als dies in lediglich um logische Auszeichnungsaspekte erweiterten graphischen Auszeichnungsystemen der Fall ist. Die Erstellung der Strukturen durch das Einfügen von Tags kann dabei in geeigneten System syntaktisch geleitet werden, um nur gewisse erwünschte Strukturen zu ermöglichen. Man spricht dann von *syntaxgesteuerten Systemen*.

Auf logischer Auszeichnung basierende Systeme können demnach in zwei Klassen aufgeteilt werden. Dies sind zunächst die Systeme, die hierarchische Strukturen mit beliebigen hierarchischen und sequentiellen Kombinationen der Tags zulassen. Zum anderen sind dies Systeme, die die möglichen hierarchischen und sequentiellen Kombinationen der Tags aufgrund vorgegebener Regeln steuern.

Es ist noch einmal zu betonen, dass es zur Überführung ausschließlich logisch ausgezeichnet erfasster Dokumente in konkrete Präsentationen aufgrund fehlender Layoutinformation im allgemeinen Fall sogenannter Stylesheets bedarf. Allgemein formuliert kann mit Hilfe einzelner in Stylesheets vorhandener Regeln für logisch ausgezeichnete Dokumentinhalte deren Verwendung bzw. Umsetzung in konkreten Repräsentationen festgelegt werden. Ein typisches Beispiel für eine Stylesheet-Regel läßt sich anhand eines Textteiles erklären, der als „wichtig“ in einem

Dokument ausgezeichnet wurde. Die Stylesheet-Regel hat in diesem Fall zur Aufgabe, die der Wichtigkeit des entsprechend ausgezeichneten Textteiles entsprechende Repräsentation in einem geeigneten System zu erzeugen. Dies wird durch die Vorgabe von Anweisungen, wie die Repräsentation zu erfolgen hat, durch das Stylesheet gewährleistet.

Eigentliche Aufgabe der logischen Auszeichnung von Dokumenten in entsprechend geeigneten System ist, die Struktur der bearbeiteten Dokumente explizit greifbar zu machen. Die Möglichkeit der logischen Auszeichnung wird in bestehenden Systemen, die dies unterstützen, aber nicht nur in diesem Sinne verwendet. Der Grund dafür liegt in einer einsichtigen Tatsache: Alle Bestandteile eines Dokumentes, die später in irgendeiner Form eine spezielle Bearbeitung benötigen, müssen über irgendeinen Mechanismus durch ein gegebenes System zugreifbar sein. Ein hierfür geeigneter Mechanismus ist aber offensichtlich die logische Auszeichnung der entsprechenden Dokumentteile.

Die logische Auszeichnungsmöglichkeit von Dokumenten wird also in aktuellen Systemen dazu benutzt, *alle* Dokumentbestandteile, die aufgrund einer beliebigen Sicht auf das Dokument zugreifbar sein müssen, kenntlich zu machen. In diesem Sinne ist die logische Struktur nur eine der möglichen Sichten auf ein Dokument.

Andere möglichen Sichten auf ein Dokument ergeben sich, indem man z.B. alle Eigennamen (wer spielt in einem Theaterstück) oder Fremdwörter (Erstellung eines Fremdwörterverzeichnisses) markiert. Somit ist klar, dass die spätere Verwendung eines Dokumentes dessen Auszeichnungsbedarf bestimmt. Unvorhergesehene gewünschte Realisierungen von konkreten Dokumenten werden somit bei unzureichender Auszeichnung eines Dokumentes zu unlösbaren Problemen führen.

2.2.5 Abstraktes Vorgehensmodell der digitalen Dokumenterstellung

In den letzten Abschnitten wurden die zwei aktuell eingesetzten Vorgehensweisen graphisches und logisches Auszeichnen zur digitalen Dokumenterstellung beschrieben. Beide Vorgehensweisen lassen sich gemeinsam in einem abstrakten Vorgehensmodell darstellen, das die schrittweise Erstellung eines Dokumentes bis zum Vorliegen einer konkreten Präsentation beschreibt.

Im Rahmen dieses Vorgehensmodells durchläuft ein Dokument mehrere Stadien. Der Erstellungprozess beginnt mit der Idee zu einem neuen Dokument, die wir mit dem Ausdruck *intendiertes Dokument* bezeichnen. Auf ihm basierend wird ein *digitales Dokument* erstellt, das in weiteren Bearbeitungsschritten von den Nutzern jederzeit geändert werden kann. Das digitale Dokument kann dann zum einen mit Hilfe eines Aufbereitungsprozesses in ein neues digitales Dokument überführt werden, das aufgrund externer Datenquellen neue Bestandteile oder z.B. durch die Erstellung eines Inhaltsverzeichnisses oder Indexes redundante Bestandteile enthält. Zum anderen kann es mit Hilfe eines Formatierungsprozesses in ein *abstraktes Bild* des Dokumentes überführt werden.

Definition 23 (Formatierprozess) *Ein Formatierprozess überführt ein digitales Dokument in ein abstraktes Bild des Dokumentes.*

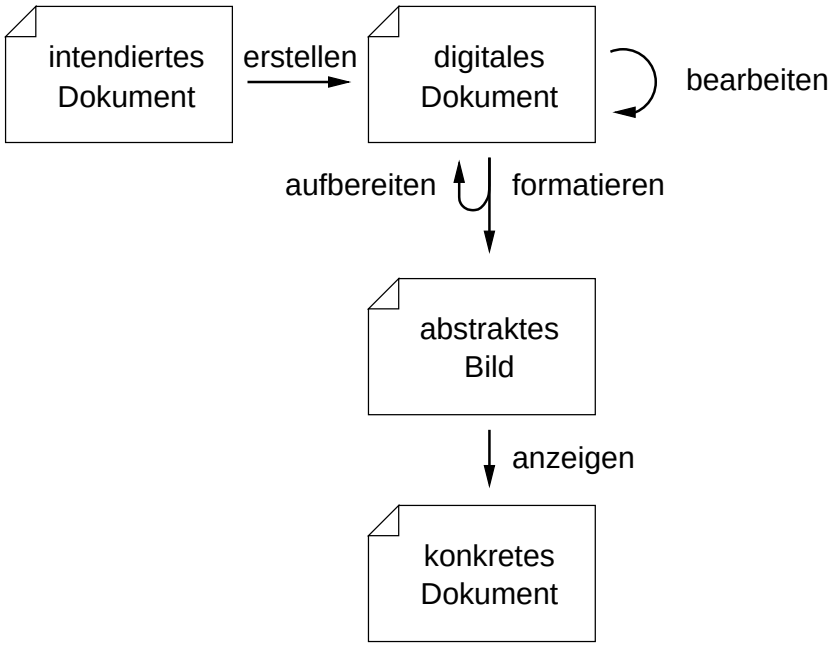

Abbildung 2.11: Abstraktes Vorgehensmodell der Dokumenterstellung

Definition 24 (Abstraktes Bild eines Dokumentes) *Das abstrakte Bild eines Dokumentes beschreibt auf nicht genauer festgelegte Art und Weise das Layout eines konkreten Dokumentes auf einem idealisierten Gerät.*

Das abstrakte Bild ist eine Zwischenform des Dokumentes auf seinem Weg zum physikalisch durch die Nutzer *greifbaren* konkreten Dokument, das mit Hilfe eines Anzeigeprozesses den Nutzern zugänglich gemacht wird. Es ist eine in gewissen Grenzen vom konkreten Ausgabemedium unabhängige Repräsentation des gegebenen Dokumentes, da es die Darstellung auf verschiedenen, untereinander *verträglichen* Präsentationsmedien erlaubt. So kann ein abstraktes Bild eines Dokumentes, das prinzipiell auf DIN A4 Papier mit Hilfe eines Druckers gedruckt werden soll, auch auf einem geeignetem Bildschirm dargestellt werden.

2.2.6 Konkretes Vorgehensmodell der digitalen Erstellung logisch ausgezeichneter Dokumente

Aufgrund der Aufgabenstellung für diese Arbeit beschränken wir uns in den folgenden Betrachtungen auf den Fall, dass die erstellten Dokumente in rein logisch ausgezeichneter Form vorliegen. Das in Abbildung 2.11 dargestellte abstrakte Vorgehensmodell läßt sich deshalb in zwei spezifische Modelle zur Verarbeitung von logisch ausgezeichneten Dokumenten abändern. Ihre Unterscheidung beruht auf dem Kriterium, ob im Modell Wissen über die Semantik der zur Auszeichnung verwendeten Tags vorliegt oder nicht.

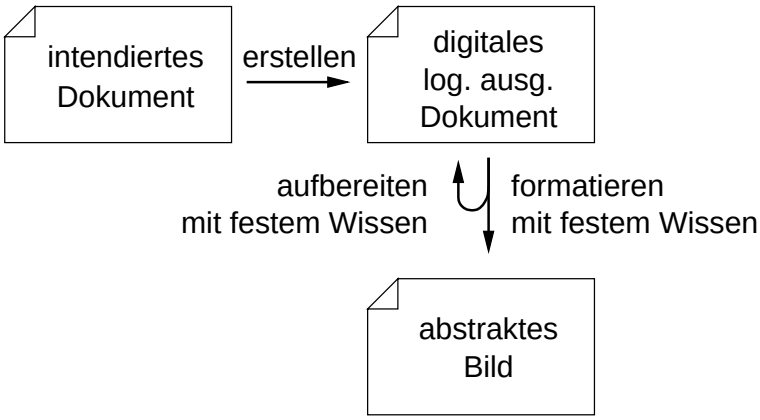

Abbildung 2.12: Konkretes Vorgehensmodell der Erstellung logisch ausgezeichneter Dokumente mit Wissen über die Semantik der verwendeten Tags

Aufbereitung mit Wissen über die Semantik der logischen Auszeichnung

Im ersten Fall setzen wir ein Softwaresystem zur Überführung des gegebenen digitalen logisch ausgezeichneten Dokumentes voraus, das Wissen über die in der logischen Auszeichnung verwendeten Tags hat. Aufgrund dieser Annahme ist es dem System möglich, Abbildungen für die einzelnen im ausgezeichneten Dokument vorkommenden logischen Objekte in ein neues digitales Dokument oder das abstrakte Bild vorzunehmen. Ein typisches Beispiel für eine derartige Architektur findet sich in den für das WWW verwendeten HTML-Browsern. Sie *verstehen* eine fest vorgegebene Menge von Tags, die sie zur Umsetzung eines gegebenen HTML-Dokumentes in eine konkrete Präsentation verwenden. Unbekannte Tags und deren Inhalt können von ihnen nicht verarbeitet werden.

Aufbereitung ohne Wissen über die Semantik der logischen Auszeichnung

Im zweiten Fall setzen wir ein Verarbeitungssystem voraus, das nicht notwendigerweise Wissen über die im logisch ausgezeichneten Dokument verwendeten Tags hat. Aufgrund fehlender semantischer Information des Systems über die Tags ist es dem System deshalb nicht möglich, eine Abbildung der im Dokument vorkommenden logischen Objekte in ein neues digitales Dokument oder das abstrakte Bild des Dokumentes vorzunehmen. Es ist notwendig, die benötigte semantische Information über die im Dokument verwendeten Tags über einen geeigneten Mechanismus zur Verfügung zu stellen. Dies geschieht in existierenden Systemen mit Hilfe sogenannter Stylesheets.

Wie in Abbildung 2.14 symbolhaft zu erkennen ist, ersetzen die Stylesheets in Systemen ohne Wissen über die logische Auszeichnung der Dokumente die fehlende benötigte semantische Information zur Aufbereitung der Dokumente.

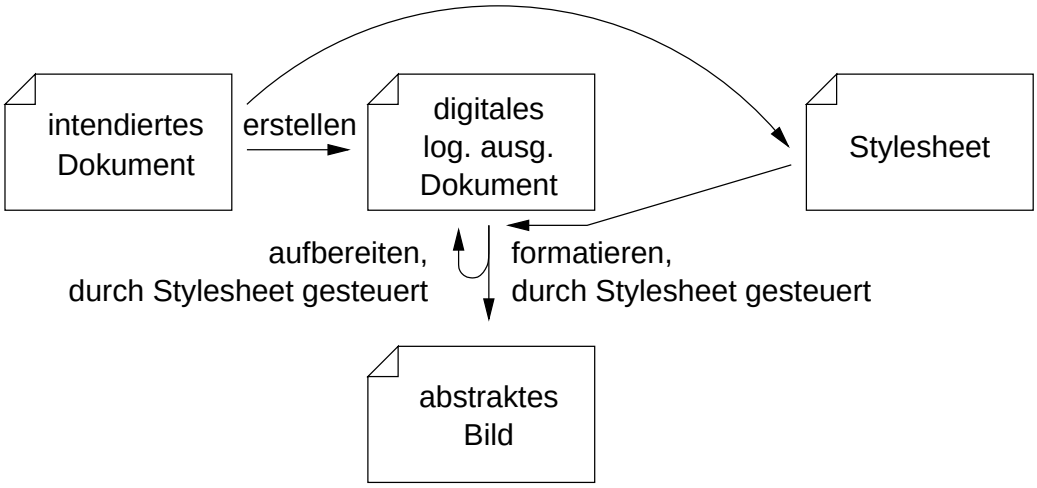

Abbildung 2.13: Konkretes Vorgehensmodell der Erstellung logisch ausgezeichneter Dokumente ohne Wissen über die Semantik der verwendeten Tags

2.3 Dokumentmodelle

In der bisherigen Arbeit wurde gezeigt, dass Dokumente im Laufe ihrer digitalen Bearbeitung mehrere Phasen durchlaufen. Dies beginnt mit dem intendierten Dokument und endet mit dem physikalisch vorliegenden konkreten Dokument. In den nächsten Abschnitten stellen wir Modelle für logisch ausgezeichnete Dokumente und deren abstrakte Bilder genauer vor.

2.3.1 Modelle für logisch ausgezeichnete Dokumente

In 2.1.2 wurde festgestellt, dass die in einem Dokument vorhandene Struktur dieses rekursiv in logische Objekte unterteilt. Durch diese Aufteilung ergibt sich die hierarchische Struktur des Dokumentes, die als Datenstruktur betrachtet ein Baum ist. Es gibt jedoch Relationen in Dokumenten, die diese Baumstruktur verletzen. Insbesondere ist dies die Verweisstruktur. Zur Repräsentation der Struktur des Dokumentes macht sie prinzipiell die Abstützung auf einen Graphen nötig.

Zur Nachbildung dieser im Vergleich zum Baum komplexeren Datenstruktur ist es in aktuellen Ansätzen üblich ein Konzept zu verwenden, das Beziehungen zwischen logischen Objekten mit Hilfe von symbolischen Namen ausdrückt. Zur gezielten Auszeichnung von logischen Objekten werden eindeutige Identifikatoren verwendet. Mit ihrer Hilfe kann man sich von beliebigen logischen Objekten auf die ausgezeichneten logischen Objekte zur Erzeugung der Verweisstruktur beziehen.

Die zwei wichtigsten Modelle, die zur logischen Auszeichnung von Dokumenten verwendet werden und auf obigen Feststellungen beruhen, sind SGML und XML. Beide Modelle werden im Folgenden überblickshaft dargestellt.

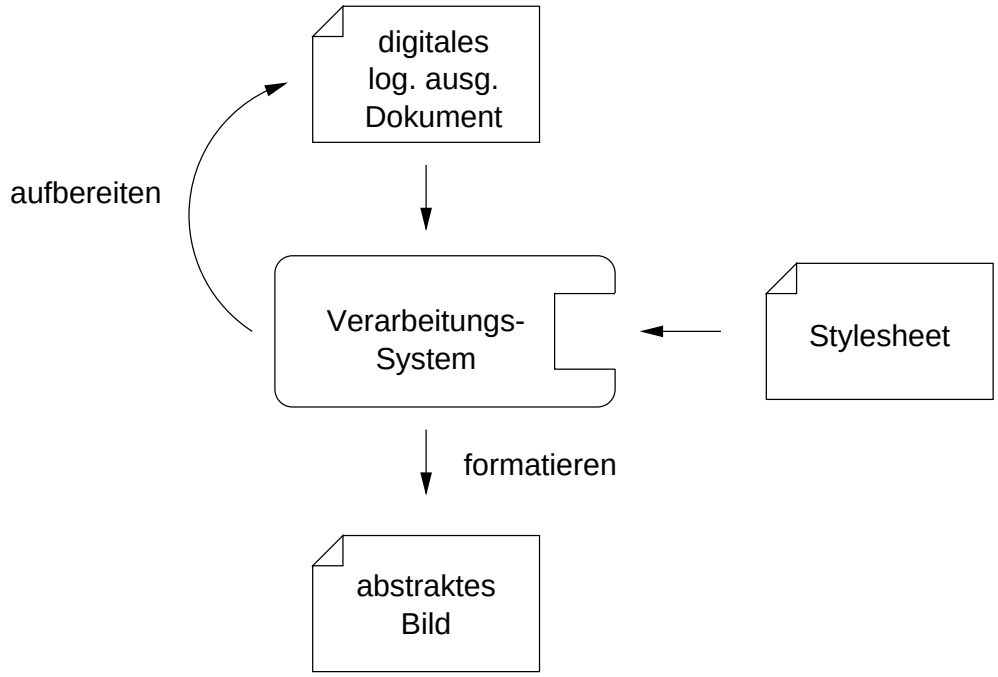

Abbildung 2.14: Verarbeitungssystem mit Stylesheetmechanismus

SGML

SGML [32, 23] (Standard Generalized Markup Language) ist ein von der ISO normiertes Konzept zur logischen Auszeichnung von Dokumenten. Ihre Entstehung geht auf die 1969 von Charles F. Goldfarb bei IBM entwickelte GML (Generalized Markup Language) zurück. SGML hat insbesondere im US-Amerikanischen Raum im Laufe der Zeit eine wesentliche Stellung im Bereich der Dokumentverarbeitung erlangt.

SGML wird oft als Sprache zur logischen Auszeichnung von Dokumenten verstanden. Tatsächlich jedoch ist SGML vom Prinzip her zuerst einmal eine *Metasprache*, mit deren Hilfe beliebige Sprachen zur logischen Auszeichnung von Dokumenten definiert werden können. Die Auszeichnungssprachen[27] werden in sogenannten *Dokument Typ Definitionen* (DTD) festgelegt, die jeweils die möglichen Strukturen einer ganzen Klasse von Dokumenten (*Dokumentklasse*) festlegen (z.B. die Klassen der Memos, Briefe und Artikel). Die momentan wohl bekannteste Auszeichnungsprache die mit Hilfe einer SGML DTD definiert ist, ist die zur Erstellung von WWW-Seiten verwendete HTML (Hypertext Markup Language).

Neben der bedeutenden Eigenschaft von SGML, eine Metasprache zu sein, legt SGML zusätzlich fest, wie die mit Hilfe der DTDs definierten Sprachen zur Auszeichnung von konkreten Dokumenten verwendet werden müssen.

Jedes SGML Dokument besteht aus zwei Teilen. Im ersten Teil wird im Wesentlichen die zum Dokument gehörige DTD formal beschrieben. Der zweite Teil ist das eigentliche mit Hilfe der in der DTD festgelegten Auszeichnungssprache logisch ausgezeichnete Dokument.

[27] Siehe auch [32].

Alle mit Hilfe einer Auszeichnungssprache erstellten Dokumente haben eine gewisse Struktur gemeinsam. So bestehen beispielsweise alle Bücher auf der obersten Gliederungsebene aus einer Folge von Kapiteln. Diese gemeinsame Struktur wird mit dem Begriff *generische Struktur* bezeichnet. Die spezielle Ausprägung der generischen Struktur in einem individuellen Dokument wird mit dem Begriff *spezifische Struktur* bezeichnet.

Die Sprache zur logischen Auszeichnung und somit die generische Struktur eines Dokumentes wird in einer DTD mit Hilfe von *Elementdeklarationen* vorgenommen. In diesen Deklarationen wird festgelegt, welche logischen Elemente in welcher Reihenfolge, Häufigkeit und Schachtelung in einem gemäß einer DTD ausgezeichnetem Dokument vorkommen dürfen bzw. müssen.

Die Elementdeklarationen führen die zur logischen Auszeichnung verwendbaren Tags ein. Eine Elementdeklaration legt dazu zum einen den Namen eines Elementes (entspricht einem logischen Objekt) fest und zum anderen die mögliche Struktur seines Inhalts (Inhaltsmodell), der, vereinfacht dargestellt, entweder Text oder weitere logische Elemente sind. Für jedes in einer Elementdeklaration eingeführte Element können weiterhin Attribute deklariert werden, die zusätzliche Informationen über das Element bereitstellen. Die DTD zu dem Dokument aus Abbildung 2.1, die keine Attribute beinhaltet, kann wie folgt aussehen:

```
<!ELEMENT  dokument      - -  (titel,abschnitt+)>
<!ELEMENT  titel         - -  (#PCDATA)>
<!ELEMENT  abschnitt     - -  (überschrift,absatz+)>
<!ELEMENT  überschrift   - -  (#PCDATA)>
<!ELEMENT  absatz        - -  (#PCDATA|hervorhebung)*>
<!ELEMENT  hervorhebung  - -  (#PCDATA)>
```

Obige DTD legt in Zeile 1 fest, dass das Element `dokument` aus einem `titel`, gefolgt von mindestens einem `abschnitt` besteht. Der Inhalt des `titel` ist gemäß Zeile 2 nicht weiter strukturiert und besteht einfach aus Text (vereinfachte Betrachtung für #PCDATA). Die restlichen Regeln sind analog zu diesen ersten beiden Regeln aufgebaut.

Die Inhaltsmodelle der Elemente werden durch *reguläre Ausdrücke* beschrieben. Sie legen fest, welche Elemente in welcher Reihenfolge und Häufigkeit in einem Element vorkommen dürfen. Zur Festlegung der Reihenfolge des Inhalts in einem Element werden in SGML drei sogenannte *Konnektoren* verwendet:

Name	Beispiel	Bedeutung
Sequenz	`(anfang , mitte , ende)`	Das mit diesem Inhaltsmodell beschriebene Element muss genau die Elemente `anfang`, `mitte` und `ende` in der gegebenen Reihenfolge beinhalten.
Oder	`(freispruch | schuldspruch)`	Das mit diesem Inhaltsmodell beschriebene Element muss genau das Element `freispruch` oder `schuldspruch` beinhalten.
Und	`(telefon & fax & email)`	Das mit diesem Inhaltsmodell beschriebene Element muss genau die Elemente `telefon`, `fax` und `email` beinhalten, wobei die Reihenfolge beliebig ist.

Zur Festlegung der Häufigkeit sind vier Konstrukte, die in SGML *occurence indicators* genannt werden, vorgesehen. Ist kein occurence indicator angegeben, so muss das Element genau einmal vorkommen.

Name	Beispiel	Bedeutung
optional (**?**)	telefon**?**	Das Element `telefon` kann einmal oder keinmal vorkommen.
benötigt und wiederholbar (**+**)	telefon**+**	Das Element `telefon` muss mindestens einmal vorkommen.
optional und wiederholbar (*****)	telefon*****	Das Element `telefon` kann beliebig oft vorkommen (auch keinmal).

Zur Auszeichnung der in einem Dokument vorkommenden logischen Objekte werden die in den Elementdeklarationen eingeführten Elementnamen verwendet. Der Start eines logischen Objektes wird durch einen Start-Tag, das Ende durch einen End-Tag gekennzeichnet. Die Syntax dieser Tags ist in SGML in einem gewissen Rahmen frei definierbar, durchgesetzt hat sich aber, ein Element `x` durch den Start-Tag `<x>` und den End-Tag `</x>` zu klammern. Das Beispieldokument aus Abbildung 2.1 könnte demgemäß wie in Abbildung 2.15, in der die DTD weggelassen wurde, aussehen:

Ein noch zu klärender Punkt in einer Elementdeklaration sind die zwei „-“-Zeichen zwischen den Elementnamen und dem Inhaltsmodell. SGML erlaubt die Auslassung von Tags in einem Dokument, falls auf ihr Vorkommen eindeutig mit Hilfe der von der Gesamtheit der in der DTD gegebenen Regeln geschlossen werden kann. Ein typisches Beispiel hierfür sind Einträge in einer Liste. Der End-Tag für die Einträge in einer Liste kann weggelassen werden, da mit dem Start eines neuen Eintrages der vorherige Eintrag beendet sein muss. Auf das Ende des letzten Eintrages kann man schließen, wenn die Liste insgesamt beendet ist. In den Elementdeklarationen würde diese Auslassungsmöglichkeit des End-Tags durch eine Ersetzung des zweiten „-“-Zeichens durch ein `O` (o für omit [engl.:auslassen]) angegeben werden. Die sich ergebende DTD für obiges Beispiel könnte dann wie folgt aussehen:

```
<dokument>
<titel>Beispiel eines Dokumentes</titel>
<abschnitt>
<überschrift>Einleitung</überschrift>
<absatz>
Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
ser diam nonummy nibh eusimod tincidunt ut laoreet
<hervorhebung>dolore magna</hervorhebung>
aliquam erat volutpat.
</absatz>
</abschnitt>
<abschnitt>
...
</abschnitt>
</dokument>
```

Abbildung 2.15: Eine mögliche SGML-Darstellung des Dokumentes aus Abbildung 2.1

```
<!ELEMENT  liste    - -  (Eintrag)+>
<!ELEMENT  eintrag  - O  (#PCDATA)>
```

Die Verweisstruktur von Dokumenten wird in SGML mit Hilfe von zu Elementen gehörigen Attributen nachgebildet. Soll in einem Dokument auf eine Abbildung Bezug genommen werden, so muss für das logische Element `abbildung` ein Attribut beispielsweise mit dem Namen `name` eingeführt werden. Für ein beliebiges Element, von dem aus auf die Abbildung Bezug genommen werden soll, muss ebenfalls ein Attribut eingeführt werden, dass z.B. `auf` heißen kann. In einer DTD sähe dies wie folgt aus:

```
<!ELEMENT  abbildung  - -   (#PCDATA) >
<!ATTLIST  abbildung  name  ID #required>
<!ELEMENT  bezug      - -   (#PCDATA) >
<!ATTLIST  bezug      auf   IDREF #required>
```

Der Wert zum Attribut `name` des Elementes `abbildung` muss, bedingt durch die Angabe des Kennwortes `ID` (SGML-Kürzel für *unique identifier*[28]) in der DTD, in einem SGML Dokument eindeutig sein und auf jeden Fall angegeben werden (Kennwort `#required`). Das Attribut `auf` des Elementes `bezug` muss als Wert einen Namen haben, der im Dokument an einer beliebigen Stelle als Wert eines `ID`-Attributes eingeführt wurde. In einem konkreten Dokument kann dies wie folgt aussehen:

```
<abbildung name="abstraktesVorgehensmodell"> ...  </Abbildung>
...
<bezug auf="abstraktesVorgehensmodell"> ...  </bezug>
```

[28] Dokumentweit eindeutiger Bezeichner.

Attribute werden nicht nur zum Aufbau der Verweisstruktur innerhalb eines Dokumentes eingesetzt, sondern unter anderem auch zur Aufnahme von Graphiken und Abbildungen in ein Dokument. Üblicherweise werden die Daten einer Graphik nicht direkt in ein Dokument aufgenommen, sondern lediglich eine Referenz z.B. auf eine Datei, in der die Graphik oder Abbildung enthalten ist. Dies kann wie folgt aussehen:

```
<bild quelle="diss/bilder/absModell.fig">
```

Ein letzter Punkt zu SGML, der hier vorgestellt wird, sind die sogenannten *Inklusionen* und *Exklusionen*. In den meisten Dokumenten gibt es Bestandteile, die praktisch an beliebiger Stelle auftreten können. Ein Beispiel hierfür sind Fußnoten. Soll die Möglichkeit dieses beliebigen Vorkommens mit Hilfe der bisherigen SGML-Konstrukte gelöst werden, ist im Allgemeinen jede der in einer DTD vorkommenden Elementdeklarationen zu ändern. Ein Ansatz, der zu unübersichtlichen und schwer wartbaren DTDs führt. Einen Ausweg bieten hier die sogenannten Inklusionen. Eine Inklusion ist eine Liste von Elementen die auf das Inhaltsmodell in einer Elementdeklaration folgt. Sie bewirkt, dass die in der Liste aufgeführten Elemente an beliebiger Stelle im Inhalt des in der Elementdeklaration deklarierten Elements auftreten können. Ein Beispiel, das Fußnoten an beliebiger Stelle in einem `abschnitt` zuläßt, sieht wie folgt aus:

```
<!ELEMENT abschnitt - - (überschrift,absatz+) +(fussnote)>
```

Die Syntax der Exklusionen ist analog zu der von Inklusionen. Allerdings wird hier anstelle des „+“-Zeichens ein „-“-Zeichen vor die Liste gestellt. Exklusionen bewirken die Aufhebung der Verwendbarkeit von durch Inklusionen zugelassenen Elementen innerhalb eines Elementes.

XML

XML 1.0 [56] (Extensible Markup Language) ist eine W3C[29] Empfehlung vom 10.2.1998, die sich auf die SGML Norm ISO 8879 abstützt. Ziel von XML ist es, logisch ausgezeichnete Dokumente beliebiger DTDs in einer derartigen Einfachheit über das WWW übertragen zu können, wie dies bisher nur mit HTML Dokumenten möglich ist [56].

Die Grundprinzipien von SGML und XML sind die gleichen:

> „XML is almost indistinguishable from SGML as practiced. XML has almost all of the capabilities of SGML that are widely supported. XML also lacks some important capabilities of SGML that primarily affect document creation, not document delivery. That’s because XML was not designed to replace SGML in every respect. As you will see, SGML remains the appropriate technology for creating and storing information.“ [4]

[29] World Wide Web Consortium.

Sowohl SGML als auch XML sind Konzepte zur Erstellung von Auszeichnungssprachen sowie logisch strukturierten Dokumenten. Ebenfalls stimmen die in beiden Konzepten zugleich verwendeten Vorgehensweisen zur Erstellung von DTDs und zur Auszeichnung von konkreten Dokumentinstanzen bis auf kleine Ausnahmen überein.

Der Unterschied zwischen SGML und XML insgesamt ist aber von wesentlicher Bedeutung. Er ergibt sich durch folgende 10 Punkte umfassende Liste, die die Designkriterien für XML umfaßt [56]:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

Durch die konsequente Anwendung obiger 10 Punkte beim Entwurf des XML Standards ergibt sich ein Verhältnis zwischen SGML und XML, das man mit dem Begriff „90/10 Regel“ charakterisieren kann: 90% aller in SGML theoretisch lösbaren Aufgaben können mit 10% des Aufwandes gelöst werden, die insgesamt zur Realisierung des SGML Standards benötigt werden. XML nimmt sich nur jener 90% der Aufgaben an und ist deshalb mit nur 10% des Aufwands, der für eine Realisierung von SGML nötig ist, zu realisieren.

Von den bei SGML vorgestellten Punkten sind in XML die folgenden Punkte nicht erlaubt: Der Und-Konnektor, die Auslassung von Tags, Inklusionen und Exklusionen. Als Ergebnis dieser Reform im XML Standard bietet sich ein vereinfachtes Modell zum Parsen der logisch ausgezeichneten Dokumente dar, das sich in einfach zu realisierenden Programmen umsetzen läßt: Eine wichtige Voraussetzung, die für eine schnelle Entwicklung im WWW-Bereich nötig ist.

Interessant ist die Tatsache, dass es in XML möglich ist, logisch ausgezeichnete Dokumente ohne eine zugehörige DTD zu verarbeiten. Einzige Voraussetzung bei dieser Art von Dokumenten ist, dass sie das sogenannte *well formed* Kriterium erfüllen: Die logische Struktur des Dokumentes muss ein Baum sein. Ist ein Dokument auch im Sprachschatz der durch eine XML-DTD erzeugbaren Sprache, so heißt es gemäß dem XML-Standard *valid*.

2.3.2 Modelle für abstrakte Bilder

Nach Definition 24 beschreibt das abstrakte Bild eines Dokumentes in einer nicht genauer festgelegten Art und Weise das Layout eines konkreten Dokumentes auf einem idealisierten Ausgabegerät.

Technisch betrachtet versteht man heute unter diesen idealisierten Ausgabegeräten die allgemeine Klasse der rasterorientierten Ausgabegeräte [engl.: raster output devices], die Geräte wie Nadeldrucker, Tintenstrahldrucker, Laserdrucker und normale Bildschirme umfaßt.

Die charakteristische Eigenschaft dieser rasterorientierten Ausgabegeräte ist, dass ausgegebene Information aus einem zweidimensionalen Vektor (Matrix) von Punkten besteht, die Pixel genannt werden. Jedes dieser Pixel kann auf den Ausgabegeräten individuell angesteuert werden. Auf einem typischen schwarz/weiß Drucker kann jedes Pixel entweder den Wert 0 oder 1 annehmen um anzuzeigen, dass das Pixel weiß bzw. schwarz ausgegeben werden soll. Als Beispiel hierzu dient Abbildung 2.16. Die Pixelmatrix auf der linken Seite dieser Abbildung korrespondiert mit dem Bild, das auf der rechten Seite der Abbildung zu sehen ist.

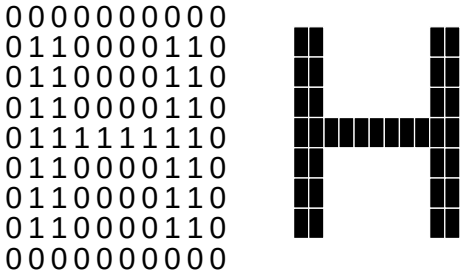

Abbildung 2.16: Pixelmatrix

Auf qualitativ besseren Geräten ist es möglich, pro Pixel nicht nur zwei verschiedene Werte speichern und ausgeben zu können. Mit Hilfe des gegebenen Wertebereiches können hier verschiedene Grauwerte bzw. Farben kodiert und dargestellt werden. Ein weiteres Qualitätsmerkmal ist neben der soeben vorgestellten *Pixel-Tiefe* noch die *Auflösung* der Geräte, die typischer Weise als Pixelanzahl/Inch angegeben wird. Sie liegt bei Bildschirmen üblicherweise zwischen 75 bis 110 Pixel/Inch, bei Tintenstrahl- und Laserdruckern im Bereich von 300 bis 1400 Pixel/Inch und schließlich bei professionellen Photobelichtern bei 2400 Pixel/Inch und mehr.

Pixelmatrizen

Die direkte Verwendung von Pixelmatrizen zur dauerhaften Repräsentation des abstrakten Bildes eines Dokumentes bringt zwei gravierende Nachteile mit sich:

- Pixelmatrizen sind sehr speicheraufwendig. So benötigt bereits eine DIN A4 Seite in schwarz/weiß Qualität bei einer Auflösung von 600 Pixel/Inch ca. 30 MB Speicher, ein Wert, der sich bei Farbdarstellung noch entsprechend der Pixel-Tiefe vervielfacht.

- Pixelmatrizen sind geräteabhängig und dies sowohl von der Pixel-Tiefe als auch von der Auflösung her.

Seitenbeschreibungssprachen

Zur Vermeidung der bei den Pixelmatrizen aufgezeigten Nachteilen werden in verfügbaren Dokumentverarbeitungssystemen sogenannte Seitenbeschreibungssprachen eingesetzt [engl.:page description language (PDL)]. Hierunter versteht man formal beschriebene Sprachen (Programmiersprachen), in denen Programme zur Beschreibung der abstrakten Bilder von Dokumenten formuliert werden können. In diesen Programmen ist es möglich, mit Hilfe von hochsprachlichen Funktionsaufrufen Ausgaben (z.B. Linien, Kreise, Buchstaben, Bilder, ...) auf abstrakte Ausgabegeräte vorzunehmen.

Die Abarbeitung eines in einer Seitenbeschreibungssprache gegebenen Programmes kann als Ausgabe eine Pixelmatrix für ein beliebiges konkretes Ausgabegerät erzeugen (siehe Abbildung 2.17). Der Prozess der Abbildung (im Englischen auch *rendering* genannt) eines abstrakten graphischen Elementes in eine Pixelmatrix wird dabei mit dem Begriff *scan conversion* bezeichnet.

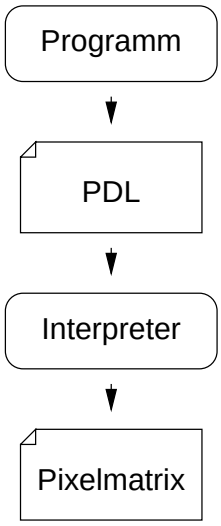

Abbildung 2.17: Darstellungsmodell für digitale Dokumente

Vorteile der Seitenbeschreibungssprache im Vergleich zu Pixelmatrizen sind also, dass zu beschreibende abstrakte Bilder sowohl geräteunabhängig, als auch wesentlich speichereffizienter beschrieben werden können.

Die zwei wichtigsten Seitenbeschreibungssprachen PostScript und PDF werden im Folgenden vorgestellt.

PostScript: PostScript [2] ist seit seiner Entwicklung im Jahre 1985 zur vermutlich meist verwendeten Seitenbeschreibungssprache geworden. Sie ist im Laufe der Jahre von Adobe regelmäßig überarbeitet worden und liegt aktuell in der dritten Version vor.

PostScript ist eine einfache interpretierte Programmiersprache mit umfassenden Möglichkeiten, graphische Ausgaben gemäß dem in Abbildung 2.17 aufgezeigten Prinzip zu beschreiben. Sämtliche graphischen Ausgaben in einem PostScript-Programm werden vom Prinzip her in eine Fläche vorgenommen, der ein Koordinatensystem unterliegt. Diese Koordinatensystem kann beliebig linear transformiert (Translation, Skalierung, Rotation, Spiegeln und Schiefstellen der Achsen) werden, wodurch beliebige nachfolgende Ausgaben den spezifizierten Transformationen unterzogen werden. Zur eigentlichen Ausgabe können dann zunächst Operatoren verwendet werden, die mit Hilfe von geraden Linien, Bögen, Rechtecken und Bézier-Kurven geometrische Figuren erzeugen. Interessant ist, dass in PostScript auch Buchstaben gemäß dieser Vorgehensweise erzeugt werden und deshalb allen möglichen Transformationen unterzogen werden können. Mit Hilfe weiterer Operatoren ist es dann möglich, diese Figuren mit Linien beliebiger Dicke und Farbe zu umgeben, sie mit einer beliebigen Farbe auszufüllen, oder sie als Clipping-Bereich zu verwenden.

Da die soeben beschriebenen graphischen Ausgabemöglichkeiten von PostScript in eine allgemeine Programmiersprache eingebettet sind, ist es Anwendungsprogrammierern möglich, die einfachen in PostScript gegebenen Grundkonstrukte zu komplexeren Operationen zusammenzufassen und als Prozeduren wiederzuverwenden.

Generell können mit einem PostScript-Programm mehrere Seiten, die z.B. auf einem Drucker ausgegeben werden sollen, beschrieben werden. Der Zugriff auf eine bestimmte Seite, die in einem PostScript-Programm beschrieben ist, muss jedoch immer sequentiell vom Beginn des Programmes an erfolgen.

PDF: Adobe hat neben PostScript eine zweite Seitenbeschreibungssprache namens Portable Document Format (PDF) [1] entwickelt und im Jahre 1993 veröffentlicht. Ziel von PDF ist es, Dokumente auf zuverlässige und einfache Art zwischen verschiedenen Programmen, Betriebssystemen und Hardware austauschen zu können.

Zur Realisierung der einfachen Austauschbarkeit beinhaltet PDF insbesondere ein Konzept, das die unabhängige Portierung eines Dokumentes auch dann gewährleistet, wenn einige der im Dokument verwendeten Schriftarten in einer Zielumgebung nicht vorhanden sind. Wie PostScript basiert PDF auf dem in Abbildung 2.17 aufgezeigten Darstellungsmodell.

Ein Dokument kann zwischen PDF und PostScript hin- und herkonvertiert werden und erzeugt in beiden Repräsentationen dieselbe Ausgabe. Die PDF-Sprache ist jedoch im Gegensatz zu PS nicht in eine allgemeine Programmiersprache eingebettet. Ein PDF Dokument kann vielmehr als eine statische Datenstruktur betrachtet werden, die für den wahlfreien Zugriff auf einzelne Seiten des abstrakten Dokumentes optimiert ist.

In Ergänzung zu PostScript kann ein PDF Dokument Daten beinhalten, die im Wesentlichen nur bei der Darstellung des Dokumentes auf einem Rechensystem zur interaktiven Nutzung von Bedeutung sind. Insbesondere sind dies Hypertext-Informationen sowie die Möglichkeit, sogenannte *Thumbnails* der einzelnen Seiten des Dokumentes im PDF Dokument zu halten.

Es ist anzumerken, dass PDF Dokumente aufgrund ihrer im Vergleich zu PostScript Dokumenten leichteren Portierbarkeit und der freien Verfügbarkeit von PDF-Dokumentbrowsern immer weitere Verbreitung finden.

2.4 Modelle für Dokumentformatierer

Seit den 60er Jahren wurden digitale Dokumentformatierer realisiert und als konkrete Produkte verfügbar gemacht [22]. Die ersten dieser Systeme waren *reine* Formatierer in dem Sinne, dass sie lediglich die physikalische Erscheinung von in einem digitalen Dokument gegebener Information behandelten (dieser Buchstabe kommt an Position (20,30) auf der aktuellen Seite, der nächste rechts daneben). In späteren Systemen wurden dann Funktionen hinzugefügt, die es gestatteten, aufgrund logischer Informationen über ein Dokument komplexere Funktionen im Dokumentformatierer anzustoßen (Erzeugung von Inhaltsverzeichnissen und Indexen, automatische Zeilennummerierung). Während in diesen Systemen der ersten und zweiten Stufe noch externe Editoren zur Erstellung der digitalen Dokumente notwendig waren, hat sich dies in den Folgeentwicklungen, die interaktive Dokumentformatierer eingeführt haben, geändert. In diesen Systemen ist zusätzlich zur Formatierfunktionalität die Möglichkeit der Dokumenterstellung integriert worden. In den folgenden Betrachtungen gehen wir jedoch nur auf die reinen Dokumentformatierer ein.

2.4.1 Aufgaben eines Dokumentformatierers

In Definition 23 haben wir festgelegt, dass die Überführung eines digitalen Dokumentes in ein abstraktes Bild mit dem Begriff Formatierprozess bezeichnet wird. Die dazu verwendete Software nennen wir *Formatierer* oder auch *Dokumentformatierer*.

Definition 25 ((Dokument)formatierer) *Eine Software, die einen Formatierprozess durchführt, heißt (Dokument)formatierer.*

Analog zu dieser Ansicht wird in [28] ein Dokumentformatierer zunächst als ein „Programm, das es einem Benutzer ermöglicht, ein druckfähiges Exemplar eines Dokumentes zu gestalten und auszugeben“, definiert. Als Anmerkung zu dieser Definition wird in [28] dann aber noch zusätzlich vermerkt: „Ein Formatierer kann auch andere Funktionen, wie z.B. Seiten- und Absatznummerierung, ausführen.“

Den Autoren von [28] ist offenbar bewußt, das ein Formatierer mehr macht, als nur den gegebenen Inhalt eines Dokumentes auf Zeilen und Seiten zu verteilen und anzuordnen, um aus einem gegebenen digitalen Dokument ein druckfähiges Dokument zu erstellen. Sie berücksichtigen in ihrer Anmerkung den Aspekt, dass in der gedruckten Form eines Dokumentes *explizit* Anteile enthalten sind, die im gegebenen digitalen Dokument nur *implizit* vorhanden sind („Absatznummerierung“). Die genaue Funktionalität, die ein Dokumentformatierer ausführt, wird von ihnen jedoch nicht festgelegt. Sie kann sich selbstverständlich von Dokumentformatierer zu Dokumentformatierer in Quantität und Qualität stark unterscheiden.

Zur Klärung dieser Funktionalität untersuchen wir im Folgenden zunächst die Kernfunktionalität von Dokumentformatierern, die sich mit der eigentlichen Druckaufbereitung beschäftigt und gehen dann auf zusätzliche Funktionen ein, die z.B. die Nummerierung von Dokumentbestandteilen zur Aufgabe haben.

Kernaufgaben von Dokumentformatierern

In [22] wird die prinzipielle Funktionalität eines Dokumentformatierers mit Hilfe der von ihm ausgeführten *Formatieroperationen* beschrieben. Formatieroperationen sind dabei nach [22] Abbildungen von in digitalen Dokumenten gegebenen *abstrakten Objekten*[30] in zweidimensional angeordnete *konkrete Objekte*[31]. In der Terminologie dieser Arbeit ergibt sich daraus folgende Definition:

Definition 26 (Formatierfunktion) *Die in einem Formatierprozess zur Abbildung von logischen Objekten in Layoutelemente verwendeten Funktionen heißen Formatierfunktionen.*

Konkrete Standardbeispiele für derartige Formatieroperationen, die in [22] gegeben werden, sind:

- Die Abbildung eines logischen Buchstabens in seine konkrete Präsentation gemäß einem gewählten Font (Glyphauswahl [engl.:character to glyph mapping]).
- Die Erzeugung eines zweidimensionalen Wortes (evtl. mit eingefügten Trennungen) aufgrund eines logischen Wortes (Trennung [engl.:hyphenation]).
- Die Verteilung des Inhalts eines logischen Paragraphens auf mehrere Zeilen (Zeilenumbruch [32] [engl.:line breaking]).
- Die Verteilung eines gesamten Dokumentes auf mehrere Seiten (Seitenumbruch [engl.:page breaking]).

Zugleich wird in [22] darauf hingewiesen, dass zwischen den einzelnen aufgeführten Formatieroperationen enge und komplizierte wechselseitige Abhängigkeiten bestehen. So kann z.B. der Zeilenumbruch Trennungen in einzelnen Wörtern nötig machen, oder der Zeilenumbruch aufgrund des Seitenumbruchs geändert werden, damit am Dokumentende nicht eine Seite entsteht, die nur eine einzelne Zeile enthält. Die Auflösung derartiger wechselseitiger Abhängigkeiten wird in aktuell verfügbaren Systemen jedoch meist nur in engen Grenzen unterstützt.

Aufgrund der oben gegebenen Beispiele und dem Studium von Texten, Tabellen, mathematischen Gleichungen und Abbildungen wird in [22] eine Menge von prinzipiellen Formatierfunktionen analysiert, die in fünf Gruppen aufgeteilt sind:

[30] Ein Beispiel für abstrakte Objekte ist gemäß [22] das Objekt **A**, das zur Klasse Buchstabe gehört und die Semantik trägt, die wir aufgrund gelerntem, allgemein anerkanntem Wissen mit dem Buchstaben „A“ verbinden.

[31] Konkrete Objekte sind nach [22] 2 dimensionale „page spaces“ (inhaltlich übersetzt:2 dimensionale Pixelmatrizen), die das formatierte Bild eines abstrakten Objektes beschreiben.

[32] Diese Funktion würde sinngemäß besser mit dem Begriff Absatzumbruch bezeichnet werden, da dies die von ihr ausgeführte Aufgabe beschreibt.

- **Funktionen zur Auswahl primitiver konkreter Objekte:**
  Üblicherweise werden in diesen Funktionen für logische Buchstaben ihre konkreten Repräsentationen aufgrund gegebener Parameter ausgewählt (Glyphauswahl).
  Im Falle von Abbildungen und Graphiken können hier aber auch spezielle Symbole und Bilder eingebunden werden.

- **Funktionen zur horizontalen und vertikalen Positionierung:**
  Funktionen dieser Gruppe werden zur *absoluten Positionierung* von konkreten Objekten auf einer Seite verwendet. Insbesondere werden durch sie Effekte wie Leerzeilen und der Abstand zwischen zwei Paragraphen realisiert.

- **Funktionen zur horizontalen und vertikalen Anordnung:**
  Hierunter sind Funktionen zusammengefaßt, die die *relative Anordnung* von konkreten Objekten bezüglich anderen konkreten Objekten bewirken. Benötigt werden diese Funktionen zur Lösung von Aufgaben wie der vertikalen Ausrichtung des Gleichheitszeichens in einer Formel oder der Zentrierung eines Tabelleneintrags.

- **Funktionen zum Umbruch:**
  Hierunter fallen Funktionen wie der Zeilen- und Seitenumbruch sowie die Einfügung von Trennungen, die konkrete Objekte gruppieren, anordnen und verteilen. (Auch werden mit Hilfe der Funktionen dieser Gruppe Aktionen ausgeführt, die nicht mehr zu den Kernaufgaben von Dokumentformatierern gehören. Insbesondere sind dies die Verwaltung von Seitenzahlen, Kopf- und Fußzeilen.)

- **Funktionen zur Skalierung:**
  Die Ausmaße gewisser Objekte müssen auf den für ihre Darstellung verfügbaren Platz angepasst werden. Dies ist z.B. nötig, wenn mehrere unterschiedlich große Bilder in einer Tabelle überblickshaft einheitlich dargestellt werden sollen. Derartige Funktionen sind in dieser Gruppe zusammengefaßt.

Brüggemann-Klein [7] führt eine ähnliche Unterscheidung von Formatierfunktionen, die Teilaufgaben im Formatierprozess übernehmen, ein. Die dort vorgeschlagenen Funktionen sind:

Funktion	Ergebnis	Aufgabe
font	Glyph	Glyphauswahl (aufgrund des im Dokument enthaltenen Textes)
lines	Zeile	Zeilenumbruch (auf Glyphen, hBoxen, vBoxen)
pages	–	Seitenumbruch (auf beliebigem Material)
hBox	hBox	Aufsammeln von Material (von Glyphen, hBoxen, vBoxen) und anordnen in horizontaler Form
vBox	vBox	Aufsammeln von Material (von Zeilen, hBoxen, vBoxen) und anordnen in vertikaler Form
attachToPage	–	Zuordnen einer Box (hBoxen, vBoxen) an eine bestimmte Stelle auf einer Seite

Erweiterte Aufgaben

Neben ihren eigentlichen Kernaufgaben übernehmen Formatierer in bestehenden Systemen in der Regel noch Aufgaben wie z.B. Seitennummerierung, Überschriftsnummerierung und die Erstellung von Verzeichnissen, Indexen sowie Kopf- und Fußzeilen. Verallgemeinert ausgedrückt geht es um die Aufgaben, die mit Hilfe von implizit in einem Dokument gegebenen Informationen, die allerdings evtl. erst während des Formatierprozesses anfallen (Seitenzahlen), neue Informationen explizit in das Dokument einfügen. Die in diesen Bereich fallenden Aufgaben können in zwei Klassen aufgeteilt werden:

- Dies sind zunächst die *statischen* Aufgaben, die ohne die zur Laufzeit eines Formatierers anfallenden Informationen auskommen. Hierunter fallen z.B. Überschriftsnummerierungen sowie die Erstellung von Verzeichnissen und Indexen.
- In die zweite Klasse fallen *dynamische* Aufgaben, die Informationen über ein Dokument voraussetzen, die erst zur Laufzeit des Formatierprozesses anfallen. Das Standardbeispiel hierfür sind Referenzen auf Seitenzahlen, deren Realisierung insbesondere bei Vorwärtsreferenzen (es wird auf Seite n eine Seite n+m referenziert) zu Problemen führen kann.

Wie bereits in 2.2.3 aufgezeigt ist die Erfüllung der hier aufgezeigten erweiterten Aufgaben bei der Formatierung von *rein* graphisch ausgezeichneten Dokumenten aufgrund der nicht gegebenen logischen Erkennbarkeit von Dokumentbestandteilen nicht möglich.

In [7] werden die erweiterten Aufgaben, die basierend auf der Strukturinformation von digitalen Dokumenten ausgeführt werden können, systematisch in drei Klassen eingeteilt:

- **Nummerierung:**
  Üblicherweise sind Überschriften in Dokumenten bezüglich ihrer Position und Hierarchiestufe nummeriert. Müsste diese Nummerierung von Hand durch die Dokumentersteller erfolgen, läge die Erstellung der Nummerierung sowie die Konsistenzprüfung der korrekten Nummerierung aller Überschriften in der Hand der Dokumentersteller. Selbige Aussage gilt auch für alle weiteren Bestandteile eines Dokumentes, die ebenfalls fortlaufend nummeriert werden müssen.
  In den meisten Dokumentformatierern sind deshalb Mechanismen integriert, die mit unterschiedlicher Mächtigkeit die Nummerierung von Dokumentbestandteilen ermöglichen.
- **Textbausteine und abgeleitete Dokumentbestandteile:**
  Textbausteine[33] sind Dokumentfragmente, die zur beliebigen Wiederverwendung in Dokumenten eingesetzt werden können. Sie werden im Dokumentformatierprozess unter anderem dazu eingesetzt, Textteile zu erzeugen, die im Eingabedokument nicht vorhanden waren. Beispiele dafür sind z.B. das Wort „*Abbildung*“, das automatisch vor Abbildungen gesetzt wird, sowie automatisch eingesetzte Briefköpfe mit einem Firmenlogo. Textbausteine können aber nicht nur fixe Bestandteile haben, sondern auch variable, aus

[33] Siehe auch [28].

dem gegebenen Dokument abgeleitete Dokumentbestandteile beinhalten. Über diesen Mechanismus können z.B. Referenzen auf die Nummern von Abbildungen oder Überschriften in gedruckten Dokumenten erzeugt werden. Auch Dokumentbestandteile wie Indexe und Inhaltverzeichnisse können über diesen Mechanismus realisiert werden.

- **Umordnung von Dokumentbestandteilen:**
  Mit Hilfe von aus Dokumenten abgeleiteten Bestandteilen können unter anderem Indexe erstellt werden. Zur Sortierung der erstellten Indexe ist es nötig, entsprechende Methoden zur Verfügung zu haben.

2.4.2 Dokumentformatierer für graphisch ausgezeichnete Dokumente

Die Eingabe der Dokumentformatierer für rein graphisch ausgezeichnete Dokumente ist prinzipiell eine syntaktisch unterscheidbare Sequenz von Daten und Tags[34] (siehe 2.2.1). In einigen Systemen können einige der Tags jedoch aufgrund von im Dokument implizit vorkommenden Eigenschaften geschlossen werden. So muss z.B. in TeX die Eingabe von Tags zur Steuerung der Zeilenumbruchfunktion nicht unbedingt erfolgen, da sie durch im Dokument vorkommende Leerzeilen gefolgert werden können.

Die Tags werden in den Dokumentformatierern als parametrisierte *Formatierkommandos* interpretiert und stoßen demgemäß die in den Dokumentformatierern vorhandenen Formatierfunktionen an. Da die in den graphisch ausgezeichneten Dokumenten vorkommenden Tags nach Voraussetzung korrekt geschachtelt sind (siehe 2.2.1), ergibt sich in der Dokumentformatiereingabe eine Blockstruktur von ineinander verschachtelten Formatierfunktionen. Entlinearisiert man diese Blockstruktur zu einem Baum, ergibt sich eine *Hierarchie von parametrisierten Formatierfunktionen*, die genau die Kernaufgaben von Dokumentformatierern übernehmen. Jeder Knoten in diesem Baum steht für eine Formatierfunktion, die mit bestimmten aus der Dokumentformatiereingabe (dem digitalen Dokument) stammenden Parametern gesteuert wird. Ein typischer derartiger Baum ist in Abbildung 2.19 dargestellt. Ein hierzu mögliches Eingabedokument ist in Abbildung 2.18 dargestellt, die daraus resultierende Ausgabe in Abbildung 2.20.

Zur Formatierung des Dokumentes werden die einzelnen Knoten des Dokumentformatiereingabebaums in einem depth-first Durchlauf angestoßen, wodurch sie die ihnen zugeordnete Formatierfunktion ausführen. Die von den Formatierfunktionen produzierten Ergebnisse werden so im Wesentlichen *bottom-up* erzeugt und können von den jeweils übergeordneten Formatierfunktionen als Eingabe verwendet werden. In der in Abbildung 2.19 gezeigten Konfiguration wird zunächst für alle logischen Buchstaben im mit (1) markierten Knoten die Formatierfunktion Glyphauswahl ausgeführt. Deren Ausgabe wird als Eingabe für die Zeilenumbruchsfunktion (Knoten (2)) verwendet, die die Buchstaben eines Paragraphen horizontal zu Zeilen anordnet und dabei evtl. Trennungen einfügt. Wie bereits vorgebracht kann über eine komplexe Interaktion, die ein enges Zusammenspiel zwischen Glyphauswahl und Zeilenumbruch erfordert, ein Worttrennungsprozess angestoßen werden, falls es ansonsten zu keinem akzeptablem Umbruch

[34] Bei Verwendung eines Tags in diesem Sinne wird er in [28] auch *eingebettetes Kommando* genannt.

```
@START:Seitenumbruch HÖHE="800" BREITE="500"
@START:Zeilenumbruch BREITE="400" AUSRICHTUNG="LINKS"
@START:Glyphauswahl FONT="TIMES" GRÖSSE="20"
  Einleitung
@ENDE:Glyphauswahl
@ENDE:Zeilenumbruch
@START:Zeilenumbruch BREITE="400" AUSRICHTUNG="LINKS"
@START:Glyphauswahl FONT="TIMES" GRÖSSE="12"
  Lorem ipsum dolor sit amet, consectetuer
  adipiscing elit, ser diam nonummy nibh
  eusimod tincidunt ut laoreet dolore magna
  aliquam erat volutpat.
@ENDE:Glyphauswahl
@ENDE:Zeilenumbruch
@ENDE:Seitenumbruch
```

Abbildung 2.18: Dokumentformatierereingabe: Der Beginn von Formatierfunktionen wird mit `@START:(Name der Formatierfunktion)` markiert, das Ende mit `@ENDE:(Name der Formatierfunktion)`, eventuelle Parameter werden hinter den Startmarkierung übergeben. Text in Zeilen ohne `@` am Anfang der Zeile wird als Eingabe für die Formatierfunktion Glyphauswahl verwendet.

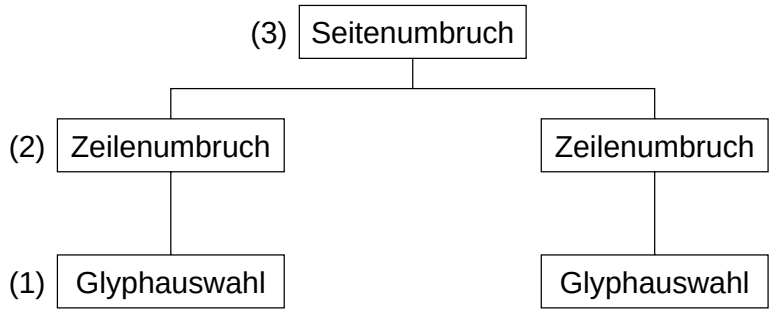

Abbildung 2.19: Formatierfunktionen-Hierarchie zum Dokument aus Abbildung 2.18

kommt. In diesem Fall ist der Informationsfluß unter den Formatierfunktionen nicht mehr rein bottom-up. Analoge Vorgänge werden im rechten Teilbaum ausgeführt. Die insgesamt derart erzeugten Zeilen werden von der Seitenumbruchsfunktion (3) vertikal angeordnet und auf Seiten verteilt.

Aufgrund der engen Interaktion zwischen den einzelnen Formatierfunktionen werden die Realisierungen der genannten Formatierfunktionen in konkreten offenen Systemen (z.B. TeX) nachweislich in einer Art vorgenommen, die man als *monolithisch* bezeichnen kann: Die einzelnen Funktionen, die die Formatierfunktionen realisieren, sind derartig miteinander verwoben, dass Änderungen in einer Formatierfunktion nur schwer vorhersehbare Nebeneffekte in anderen Formatierfunktionen erzielen können. Als Konsequenz daraus sind derartige Systeme kaum und nur von Experten zu warten bzw. zu ergänzen [7].

Dokumentformatierer für rein graphisch ausgezeichnete Dokumente sind nicht in der Lage, erweiterte Aufgaben eines Dokumentformatierers auszuführen. Ihnen fehlt hierzu das Wissen

Einleitung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, ser diam nonummy nibh eusimod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Abbildung 2.20: Mögliche Ausgabe zum Dokument aus Abbildung 2.18

um die logische Aufgabe der Bestandteile ihrer Eingabe.

Ein Datenmodell für Dokumentformatierer

Dokumentformatierer setzen aufwendig gestaltete Seiten zusammen, indem sie zunächst die einzelnen Buchstaben eines Dokumentes zu Zeilen anordnen, anschließend die Zeilen zu Paragraphen anordnen, und diese dann auf Seiten verteilen. Vom Konzept her werden also Objekte zu neuen Objekten zusammengefaßt, die wiederum zu noch größeren Objekten zusammengefaßt werden, u.s.w.. Ein mögliches Datenmodell, das diesem Prozess zugrundeliegen kann, wurde 1979 von D.E. Knuth für das TeX Dokumentverarbeitungssystem entwickelt und *Box and Glue* genannt [34]. In Varianten hat dieses Modell Einzug in viele Dokumentformatierer gefunden.

TeX's *Boxen* sind eine Abstraktion für etwas, das im Rahmen eines Formatierprozesses gesetzt werden soll. Sie sind zweidimensionale *Dinge* [35] mit rechteckiger Form, die wie in Abbildung 2.21 gezeigt eine Höhe, Tiefe und Breite haben. Die Grundlinie einer Box legt den relativen Bezugspunkt für die Höhe und Tiefe der Box fest, der Referenzpunkt[35] und der in TeX nicht explizit erwähnte Fluchtpunkt[36] dienen zur Anordnung von Boxen.

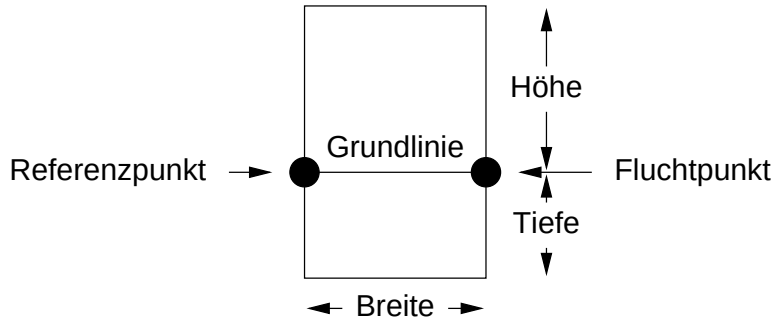

Abbildung 2.21: Eine Box im Box and Glue Modell von TeX

Bereits jeder Buchstabe ist für TeX eine Box, die mit Hilfe einer Formatierfunktion zur Glyphauswahl erzeugt wird. Ein Designer legt fest, welche Ausmaße ein gewisses Zeichen in einem Font hat und welche Pixelmatrix zur Darstellung für den durch die Box repräsentierten Buchstaben verwendet werden soll.

[35] In [30] auch Position Point genannt.

[36] Der Fluchtpunkt [engl.:Escapement Point] wird explizit z.B. in [30] eingeführt.

Die derart erzeugten Buchstaben werden horizontal zu neuen Boxen gruppiert, um z.B. ganze Wörter zu repräsentieren. Der Referenzpunkt eines Folgebuchstabens wird dabei in der Regel auf den Fluchtpunkt der Box des vorhergehenden Buchstabens positioniert (siehe Abbildung 2.22 a). In einigen Ausnahmefällen werden Boxen jedoch aus ästhetischen Gründen überlappend oder mit etwas zusätzlichem Abstand angeordnet, was *Kerning*[37] genannt wird. Als Beispiel hierzu dient Abbildung 2.22 b) und c). Weiterhin werden im Rahmen dieser Anordnung in qualitativ hochwertigen Formatierern unter gewissen Bedingungen mehrere Buchstaben Boxen durch eine neue Box ersetzt[38], die *Ligatur* genannt wird. Informationen die zur Durchführung des Kernings als auch der Ligaturbildung benötigt werden, sind von den Font-Designern den Formatierfunktionen zur Verfügung zu stellen.

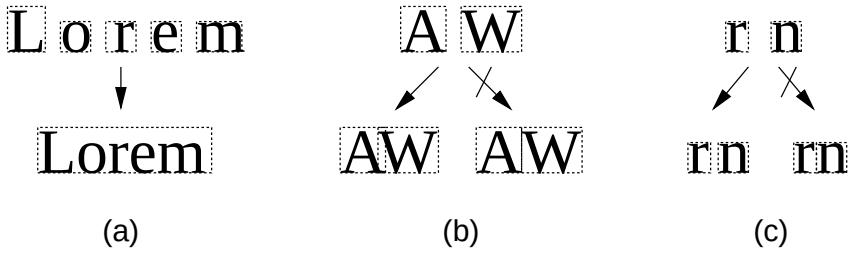

Abbildung 2.22: Boxgruppierung in TeX: Zur Anordnung eines Wortes aus Buchstaben werden horizontal die Referenzpunkt der Folgebuchstaben auf die Fluchtpunkte der Vorgängerbuchstaben gesetzt und die Grundlinien vertikal gleichgesetzt (a).
Zur Durchführung des Kernings werden die Boxen für die Buchstaben A und W überlappend angeordnet (b), während die Boxen für r und n etwas auseinandergezogen werden (c), damit das gedruckte Ergebnis nicht mit einem m (rn anstatt m) zu verwechseln ist.

Leerzeichen werden in dem Box and Glue Modell von TeX nicht einfach durch normale Boxen realisiert, für sie wird der sogenannte *Glue* verwendet. Glue (siehe Abbildung 2.23) ist ein *Ding* das wie eine Box bei Verwendung in einer horizontalen Anordnung eine bestimmte Breite hat. Zusätzlich zu dieser Breite hat ein Glue zwei Werte die angeben, um wieviel der Glue gestaucht (Stauchbarkeit) bzw. gedehnt (Dehnbarkeit) werden kann. Konzeptuell hat ein Glue wie auch schon die Boxen einen Referenz- und einen Fluchtpunkt. Die genaue Position des Fluchtpunkts eines Glues und damit die tatsächliche Breite eines Glues wird erst durch die Formatierfunktion festgelegt, die den Glue verarbeitet. Mögliche Werte des Fluchtpunktes sind durch den Fluchtpunktbereich festgelegt.

Dieses Konzept des Glues wird z.B. benötigt, um die Wörter eines Paragraphen in Blocksatz zu setzen, in dem alle Zeilen die gleiche Länge haben. Reicht in einer Zeile der Platz nicht mehr ganz für ein zusätzliches Wort, so können alle Glues etwas gestaucht (Abbildung 2.24 b)) und das Wort in die Zeile aufgenommen werden bzw. das Wort in die nächste Zeile geschoben und alle Glues etwas gedehnt werden (Abbildung 2.24 c)). Für den Fall das weder die eine, noch die andere Lösung möglich ist, kann noch versucht werden, das betroffene Wort zu trennen.

[37] Auch Unterschneiden.

[38] Z.B. die Ersetzung der zwei aufeinanderfolgenden Buchstaben f und i durch die Ligatur fi, anstelle der beiden Einzelzeichen fi.

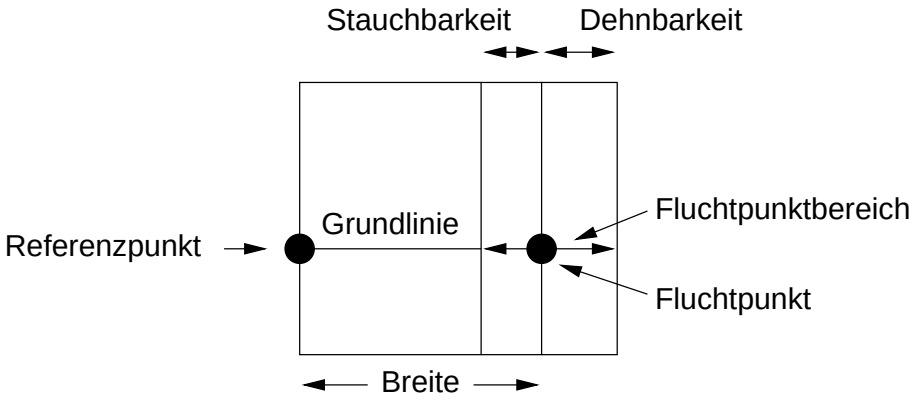

Abbildung 2.23: Ein Glue im Box and Glue Modell von TeX

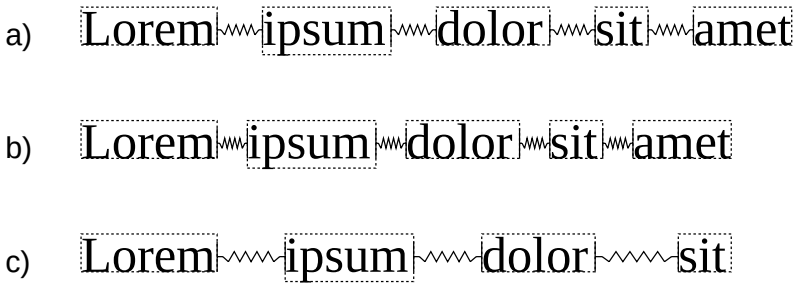

Abbildung 2.24: Aspekte des Zeilenumbruchs im Box and Glue Modell: In Zeile a) ist eine Wortfolge mit der Standardbreite der in ihr enthaltenen Glues abgebildet. Zeile b) zeigt dieselbe Wortfolge mit gestauchten Glues für eine bestimme Zeilenbreite, Zeile c) eine Variante mit gedehnten Glues, in der das letzte Wort herausgefallen ist.

Weiterhin werden Glues auch in vertikalen Anordnungen verwendet. In diesem Fall hat der Glue keine Breite sondern eine Höhe, die Stauch- und Dehnbarkeit beziehen sich dann auf die vertikale Ausdehnung des Glues.

2.4.3 Dokumentformatierer für logisch ausgezeichnete Dokumente

Die primäre Eingabe der Dokumentformatierer für logisch ausgezeichnete Dokumente sind in digitaler Form vorliegende logisch ausgezeichnete Dokumente. Beachtet man, dass das Ziel der logisch ausgezeichneten Dokumente eine vollkommene Trennung der Daten und Struktur eines Dokumentes von jeglichen auf ihnen auszuführenden Verarbeitungsaktionen ist, stellt sich die Frage, wie derartige Dokumente verarbeitet und insbesondere formatiert werden können. Wie in 2.2.6 gezeigt müssen Mechanismen zur Verfügung gestellt, die die auszuführenden Verarbeitungsaktionen mit den logisch ausgezeichneten Dokumenten wieder in Verbindung bringen. Im Folgenden werden zwei mögliche Vorgehensweisen vorgestellt.

Dokumentformatierer mit Wissen über die Semantik der Auszeichnung

Die in 2.2.6 vorgestellte Verarbeitung mit Wissen über die Semantik der logischen Auszeichnung ist in Bezug auf die Kernaufgaben von Dokumentformatierern im Wesentlichen gleichwertig zu den Dokumentformatierern für graphisch ausgezeichnete Dokumente. Aus diesem Grund betrachten wir diese Systeme im Folgenden nur kurz.

Die Verbindung zwischen Dokument und den auszuführenden Verarbeitungsaktionen wird in diesem Fall durch fest dem Verarbeitungssystem vorgegebenes Wissen realisiert. Die logischen Tags können schematisch betrachtet aufgrund des vorhandenen Wissens über die Tags als parametrisierte Formatierkommandos (siehe 2.4.2) interpretiert werden und demgemäß den Aufbau einer Hierarchie von parametrisierten Formatierfunktionen steuern.

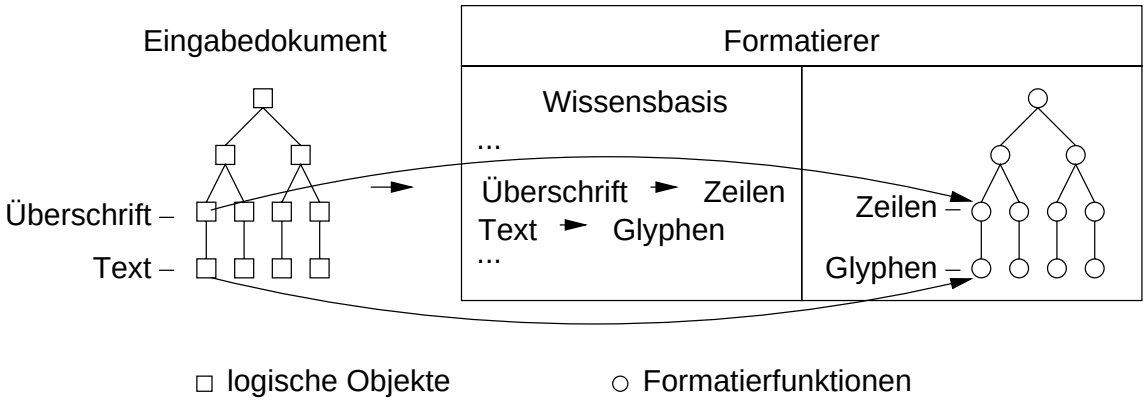

Abbildung 2.25: Dokumentverarbeitung in einem Dokumentformatierer mit festen Verarbeitungsvorschriften: Den im logisch ausgezeichneten Eingabedokument vorkommenden logischen Objekten werden aufgrund einer fest im Formatierer vorgegebenen Wissensbasis Formatierfunktionen zugewiesen.

Aufgrund des Wissens über die Semantik der zu verarbeitenden Tags ist es im Vergleich zu Dokumentformatierern für graphisch ausgezeichnete Dokumente in diesem Fall für die Dokumentformatierer aber zusätzlich möglich, auch erweiterte Aufgaben auszuführen. Realisiert wird dies, indem fest im System vorgegebene Aktionen ausgeführt werden, um z.B. Referenzen auf Seitenzahlen umzusetzen oder ein Inhaltsverzeichnis zu erzeugen.

Diese Art von Dokumentformatierern mit fest vorgegebenem Wissen ist unflexibel in Bezug auf die Art der verarbeitbaren Dokumente (es können nur Dokumente verarbeitet werden, die eine a priori bekannte Menge von Tags beinhalten) als auch in Bezug auf die mögliche Verarbeitung an sich (es können nur fest vorgegebene Verarbeitungsmöglichkeiten ausgewählt werden).

Dokumentformatierer ohne Wissen über die Semantik der Auszeichnung

Das zweite in 2.2.6 vorgestellte Verarbeitungsmodell gleicht das fehlende Wissen über die Semantik der zur logischen Auszeichnung verwendeten Tags mit Hilfe externer Information, die

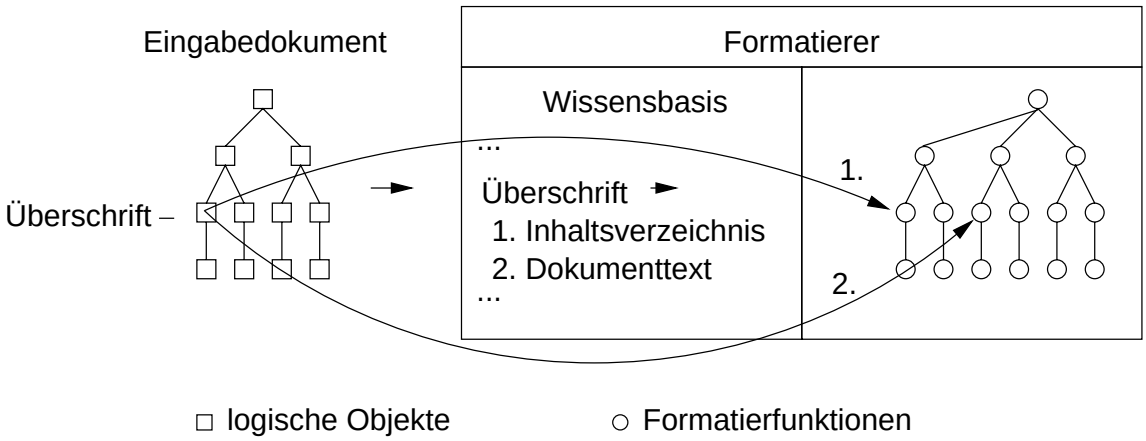

Abbildung 2.26: Doppelte Verwendung von Daten und Struktur des Eingabedokumentes: Der Inhalt der Überschrift wird aufgrund einer fest vorgegebenen Regel im Formatierfunktionen Baum doppelt zur Erstellung einer Formatierfunktion verwendet.

in sogenannten Stylesheets zur Verfügung gestellt wird, aus. Die Determiniertheit der zuletzt vorgestellten Dokumentformatierer mit Wissen über die Semantik der Tags wird auf diese Weise auf jeden Fall zumindest in Bezug auf die Menge der verarbeitbaren Tags umgangen. Die primäre Eingabe für den Dokumentformatierer bleibt das logisch ausgezeichnete Dokument.

Aktuell verfügbare Systeme arbeiten wie folgt: Das logisch ausgezeichnete Eingabedokument besteht aus einer syntaktisch unterscheidbaren Sequenz von Daten und Tags, mit deren Hilfe die logische Struktur des Dokumentes im Dokumentformatierer in einen Strukturbaum (logische Struktur) überführt werden kann. Das Stylesheet beinhaltet für die in dem Strukturbaum vorkommenden logischen Objekte Regeln die vorgeben, wie die jeweiligen Knoten im Dokumentformatierer verarbeitet werden sollen. Aktuell verfügbare Systeme unterscheiden sich dabei in Bezug auf den Funktionsumfang der möglichen Verarbeitung extrem. Nach [7] reicht das Spektrum von Systemen die mit *reiner Substitution* arbeiten bis hin zu *voll programmierbaren* Systemen.

Im einfachsten Fall, der die Systeme umfaßt, die nur die reine Substitution unterstützen, werden die logischen Knoten des Dokumentstrukturbaums aufgrund der Regeln im Stylesheet durch fest zur Verfügung stehende parametrisierte Formatierfunktionen ersetzt. Das Stylesheet spezifiziert somit eine Abbildung vom logischen Strukturbaum in eine Hierarchie von parametrisierten Formatierfunktionen. In diesem Fall dient das Stylesheet nur als Möglichkeit, die im logisch ausgezeichneten Dokument fehlende Layoutinformation zu ersetzen. Das Stylesheet unterstützt also nur die Verwendung der Kernfunktionen des Dokumentformatierers.

Im komplexeren Fall von voll programmierbaren Systemen dienen die in den einzelnen Regeln des Stylesheets gegebenen Verarbeitungsvorschriften nicht nur einer einfachen Abbildung des logischen Strukturbaumes in eine Hierarchie von Formatierfunktionen, was wieder einer Substitution der logischen Knoten durch die Formatierfunktionen entspricht (obgleich dieser Fall als triviales Vorgehen durchaus möglich ist.). Vielmehr werden die in den Regeln zur Verfügung gestellten Verarbeitungsvorschriften als Programme betrachtet, die das Eingabedokument beliebig verarbeiten können, um eine Ausgabe beliebiger Art zu erzeugen (z.B. eine

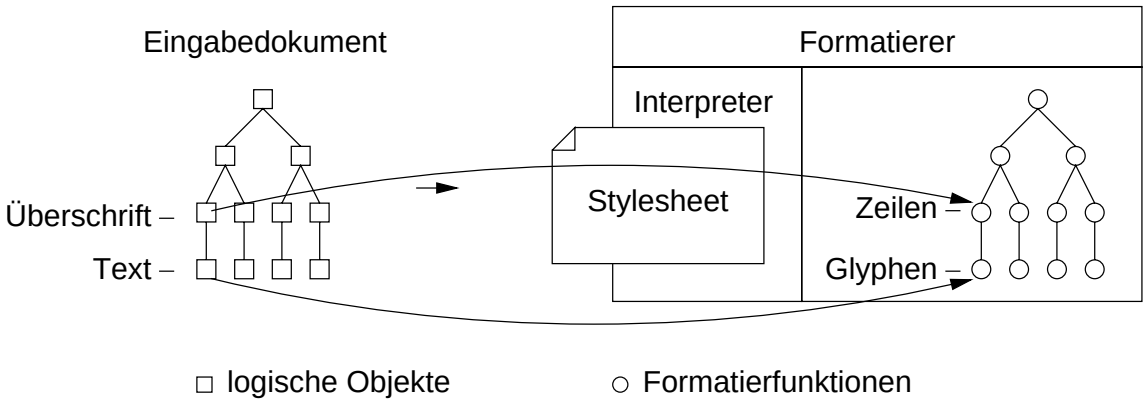

Abbildung 2.27: Dokumentverarbeitung in einem durch Stylesheets gesteuerten Dokumentformatierer

Hierachie von Formatierfunktionen oder unüblicher eine Statistik über die im Dokument enthaltenen Buchstaben). Im Regelfall wird im Dokumentformatierer mit Hilfe der in den Regeln vorgegebenen Verarbeitungsvorschriften aufgrund des gegebenen logischen Dokumentstrukturbaumes ein *neuer* Strukturbaum von Formatierfunktionen aufgebaut. Dieser muss nicht mehr wie im einfachen Substitutionsfall dieselbe Struktur wie der Ausgangsbaum haben. Er kann statt dessen um Teilbäume ergänzt sein, die ein Inhaltsverzeichnis oder einen Index repräsentieren oder Teilbäume umgeordnet beinhalten, um eine alphabetische Sortierung eines Schlagwortverzeichnisses zu realisieren. In voll programmierbaren Systemen können also auch die erweiterten Aufgaben von Dokumentformatierern Stylesheet-gesteuert realisiert werden.

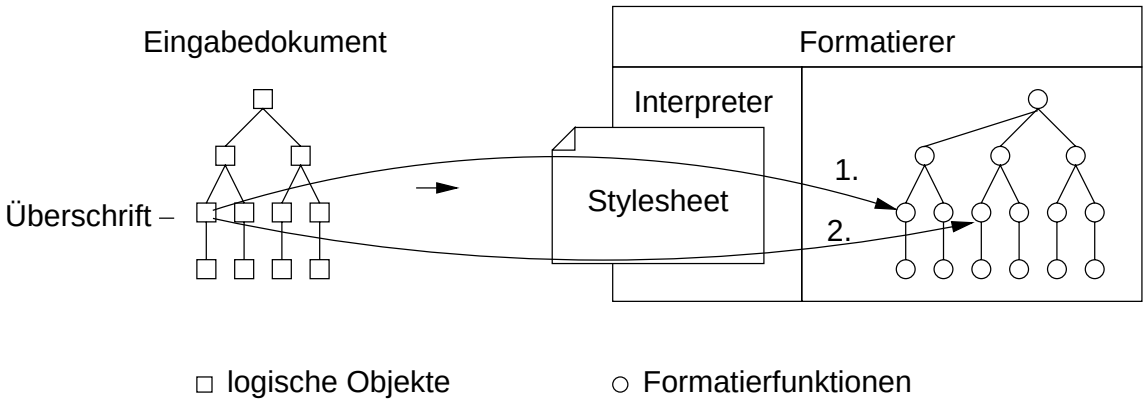

Abbildung 2.28: Durch ein Stylesheet gesteuerte Mehrfachverwendung von logischen Objekten in einem Dokumentformatierer

2.5 Stylesheet Modelle

Am Beispiel der Modelle für Dokumentformatierer hat sich gezeigt, dass Architekturen, die von außen mit Hilfe von Stylesheets gesteuert werden können, wesentlich flexibler sind, als Systeme

mit *fest verdrahteten* Verarbeitungsvorschriften. Aus diesem Grund haben sich in existierenden Dokumentformatierern und Dokumentverarbeitungssystemen Stylesheets zur Spezifikation durchgesetzt. Um einen tieferen Einblick in die Aufgabengebiete und Möglichkeiten der Stylesheets zu erhalten, untersuchen wir deshalb zunächst in der Literatur vorliegende Definitionen zum Begriff Stylesheet. Im Anschluss daran werden zwei ausgesuchte Stylesheetansätze bestehender Systeme vorgestellt. Abschließend betrachten wir aktuelle Ansätze, die in Form von internationalen Standards und allgemein anerkannten Vorschlägen des W3C vorliegen.

2.5.1 Stylesheets für Dokumentformatierer in der Literatur

Unter [58] wird vom W3C, der gegenwärtig aktivsten Gruppe im Bereich der Stylesheetsprachen, der grundlegende Charakter von Sytlesheets wie folgt festgelegt: „Style Sheets describe how documents are presented on screens, in print, or perhaps how they are pronounced."

In der SGML Norm [32] wird ein Stylesheet[39] sehr offen als „a description of the processing that is required", bezeichnet. Ein Stylesheet kann gemäß [23] zum einen in einer natürlichen Sprache vorliegen und Anweisungen für einen Setzer enthalten, wie ein gegebenes logisch ausgezeichnetes Dokument zu setzen ist (siehe zu diesem Vorgang auch [8]). Zum anderen wird hier die Möglichkeit aufgezeigt, dass ein Stylesheet eine formale Spezifikation sein kann, die von einem Interpreter verarbeitet werden kann, um konkrete Prozeduren für ein Verarbeitungssystem zu erzeugen. Die in [23] getroffenen Aussagen sind auf einem sehr abstrakten Niveau und lassen einen weiten Interpretationsspielraum zu, welche funktionalen Bereiche in einem Dokumentverarbeitungssystem mit ihrer Hilfe gesteuert werden können. Diese weite Auffassung ist, wie sich in einem der vorgestellten Stylesheetansätze zeigen wird, in modernen komplexen Systemen eine unabdingbare Notwendigkeit.

In dem Übersichtsartikel *Styles in Document Editing Systems* [33] werden Stylesheets funktional enger als in [23] festgelegt, indem ihre Aufgaben genauer eingeführt werden. Ein Stylesheet wird hier als „a collection of Style Rules intended to be used together" definiert, das entweder separat vom Dokument verwaltet werden kann, oder mit diesem unmittelbar verbunden ist. Notwendigerweise werden nach dieser Definition in [33] auch *Style Rules* festgelegt: Eine Style Rule ist „a named collection of settings for properties of document content objects".

Die in [33] aufgeführten Beispiele zeigen, dass es den Autoren bei der Verwendung von Stylesheets im Wesentlichen um die Zuordnung von Formatierfunktionen (Kernfunktionalität von Dokumentformatierern) zu den logischen Objekten des Dokumentes geht und weniger um die erweiterten Aufgaben der Dokumentformatierer.

In [33] wird die Möglichkeit, eine gewünschte Änderung in einem Dokument mit Hilfe eines Dokumentverarbeitungssystems konsequent zu realisieren, *dynamische Funktionalität* des Systems genannt. Ein Beispiel für die dynamische Funktionalität eines Systems ist z.B. die Möglichkeit, konsequent die Schriftgröße für alle Kapitelüberschriften in einem Dokument zentral an einer Stelle zu bestimmen. Allgemein wird die dynamische Funktionalität eines Systems

[39] In [23] wird vorgeschlagen, den Begriff Stylesheet mit dem Ausdruck „specification sheet" zu bezeichnen, da mit Hilfe von Stylesheets *nicht nur* der Vorgang des „conventional formatting" beschrieben wird.

benutzt, um die *statische Funktionalität*[40] eines Systems zu steuern. Die dynamische Funktionalität ist der zentrale Gesichtspunkt [33], weshalb die Dokumenterstellung in modernen Dokumentverarbeitungssystemen wesentlich schneller, konsistenter und damit auch sicherer vor sich geht als in Systemen ohne diese Funktionalität (z.B. Systeme für graphisch ausgezeichnete Dokumente, bei denen man eine konsistente Änderung von Hand realisieren muss). Beachtet man, dass Stylesheets genau diese Funktionalität steuerbar machen, erkennt man sofort ihre zentrale Bedeutung im Bereich der Dokumentverarbeitung.

Interessant ist in [33] die Einführung von *Style Environments*, die als „set of style rules available to be referenced in a document“ bezeichnet werden, wobei ergänzt wird: „One or more stylesheets may contribute to a documents style environment“. Das vollständige Modell, das sich aufgrund der Einführung der drei Einheiten Style Environment, Stylesheet und Style Rule ergibt, ist hierarchisch in drei Stufen gegliedert und dadurch sehr offen: einzelne Style Rules beschreiben die Verarbeitung eines kleinen Aspekts eines Dokumentes (z.B. einer Überschriftsebene) und können aufgrund inhaltlicher Gesichtspunkte zu Gruppen, den Stylesheets, zusammengefaßt werden, die einen gewissen größeren Aspekt (z.B. die Verarbeitung aller Überschriftsebenen) der Verarbeitung eines Dokumentes beschreiben. Mehrere dieser Gruppen (Stylesheets für Überschriften, Fußnoten, Listen, ...) können dann gemeinsam in einem Design Environment zur Spezifikation der Verarbeitung eines gesamten Dokumentes bedarfsgerecht zusammengestellt und verwendet werden.

Die in [33] vorgenommene an sich natürliche Aufteilung eines Stylesheets in Style Rules wird auch in [28] formalisiert. Ein Stylesheet wird hier *Druckformatvorlage* genannt und als „Zusammenstellung von gespeicherten Druckformaten in einer Datei, die das Layout des Dokumentes bestimmen, dem sie zugeordnet sind“, beschrieben. Ein *Druckformat* wird als eine „benannte Menge von Formatierungs-Befehlen, die es einem Benutzer ermöglicht, viele Attribute gleichzeitig auf Text anzuwenden und das Layout eines Dokumentes durch Anwendung der gleichen Formatierungsmerkmale auf verschiedene Textteile zu vereinheitlichen“ festgelegt. Aufgrund der nicht genauer festgelegten Möglichkeiten, die in der Definition zur Druckformatvorlage mit dem Begriff Layout verbunden werden, ist es dem Leser freigestellt zu entscheiden, ob bei dieser Definition der Stylesheets auch die Möglichkeiten der erweiterten Aufgaben eines Dokumentformatierers mit abgedeckt sind.

In [7] werden sehr genaue Aussagen bezüglich eines Stylesheet Modells getroffen. Ein Stylesheet ist hier eine Tabelle mit einem Eintrag für jedes logische Objekt, das in einem Dokument vorkommen kann. Die Einträge werden gemäß [33] Style Rule genannt und können mehrere sogenannte *Transcriptions* enthalten, die gesteuert über Kontextprädikate von einem Interpreter ausgeführt werden können. Der Wert eines Kontextprädikates kann basierend auf der logischen Struktur eines Dokumentes berechnet werden. Das dort vorgestellte System transformiert aufgrund der im Stylesheet gegebenen Transcriptions ein zu verarbeitendes logisch ausgezeichnetes Dokument in eine Hierarchie von Formatierfunktionen, indem es zunächst mit Hilfe von Nummerierungen, Textbausteinen, abgeleiteten Dokumentbestandteilen und Umordnungen die nötigen Anreicherungen des Ausgangsdokument vornimmt und dann den logische Knoten

[40] Die statische Funktionalität eines Systems ist im Sinne von [33] die Menge der verfügbaren Formatierfunktionen.

parametrisierte Formatierfunktionen zuweist. Der hier gewählte Ansatz unterstützt die Steuerung sowohl der Kernaufgaben als auch der erweiterten Aufgaben eines Dokumentformatierers mit der Hilfe von Stylesheets.

Besonderer Wert wird in [7] auf die Beachtung der Kontextprädikate gelegt. Es wird angeführt, dass allgemein anerkannte Designregeln zwar vorschlagen, alle logischen Objekte einer Art gleich zu behandeln, zu diesen Regeln aber anerkanntermaßen Ausnahmen bestehen. Diese Ausnahmen ergeben sich nach [7] aufgrund von Traditionen in der Satzkultur, ästhetischen Anforderungen oder dem strukturellen Kontext eines logischen Objektes im Dokument.

Als Resümee aus den hier untersuchten Arbeiten über Stylesheets kann man festhalten, dass Stylesheets im Bereich der Dokumentverarbeitungssysteme eine zentrale Rolle einnehmen: Erst sie ermöglichen in entsprechenden Systemen die flexible Steuerung der Anreicherung eines Dokumentes um Komponenten wie beispielsweise Verzeichnisse sowie der dynamischen Funktionalität von Dokumentverarbeitungssystemen und somit eine vereinfachte Verwendung der statischen Funktionalität von Dokumentverarbeitungssystemen. Stylesheets spezifizieren im Rahmen eines geeigneten Systems die Verarbeitung, der ein gegebenes logisch ausgezeichnetes Dokument unterzogen werden soll. Beziehen sich die Auswirkungen der Stylesheets in den meisten Betrachtungen noch auf die Steuerung der Kernfunktionalität der Formatierer, so gehen einige Ansätze weiter und beziehen auch schon die erweiterte Funktionalität der Dokumentformatierer mit in den Steuerbereich der Stylesheets ein. Die den verschiedenen Ansätzen zugrundegelegten nicht explizit vorgestellten Verarbeitungsmodelle sind in allen Fällen vom Prinzip her identisch: Logisch strukturierte Dokumente in Baumform werden in eine Hierarchie von parametrisierten Formatierfunktionen überführt. Die Verarbeitungsmodelle decken im Wesentlichen nur den Bereich der Kernfunktionalität von Dokumentformatierern ab und gehen, falls überhaupt, nur informell auf die Ergänzungen für die erweiterte Funktionalität ein.

2.5.2 Stylesheets in bestehenden Systemen

Stylesheets in typischen Büro Dokumentverarbeitungssystemen

Stellvertretend für die Klasse der kommerziellen Textverarbeitungssysteme im Bürobereich soll das Textverarbeitungssystem Microsoft Word vorgestellt werden. Der hier realisierte Stylesheet Ansatz ist in Bezug auf Qualität und Quantität der angebotenen Funktionalität vergleichbar mit anderen am Markt verfügbaren Systemen. Die schematischen Ergebnisse der Untersuchung sind im Folgenden zusammengestellt.

Ein Dokument in Word besteht im Wesentlichen aus einer Sequenz von Absätzen die Text beinhalten. Die einzelnen Absätze können mit Hilfe eines angebotenen Mechanismus einer logischen Auszeichnungsklasse zugeordnet werden. Möglich sind hier insbesondere fest vorgegebene Kategorien wie Überschriften für bestimmte logische Ebenen und *normale* Absätze. Es ist aber auch möglich, eigene logische Auszeichnungsklassen zu definieren und für sie festzulegen, ob sie von ihrer Aufgabe her z.B. eine Überschrift für eine bestimmte Ebene darstellen. Analog zu dieser Auszeichnungsmöglichkeit können ebenfalls einzelne Textpassagen des Dokumentes mit Hilfe von fest vorgegebenen und frei definierbaren Klassen logisch ausgezeichnet werden.

Mit Hilfe eines Stylesheetmechanismus kann für die in einem derart gegebenen logisch ausgezeichneten Dokument vorkommenden Bestandteile ihre Verarbeitung festgelegt werden. Dies geschieht mit Hilfe von Style Regeln, die getrennt für Absätze und für Textblöcke erstellt werden können. Für jede in einem Dokument vorkommende logische Absatz- oder Textklasse gibt es eine Regel im Stylesheet zu diesem Dokument. Jede dieser Regeln spezifiziert *ausschließlich* Parameter, die von Formatierfunktionen zur Formatierung des Dokumentes verwendet werden. Die anzuwendenden Formatierfunktionen können in Style Regeln nicht angegeben werden, sie sind implizit schon durch das Dokument festgelegt. So wird automatisch auf den Inhalt eines jeden logischen Paragraphen und jeder Überschrift eine Zeilenumbruchsfunktion angewendet, auf den Text der in den Absätzen vorkommenden Textblöcke eine Glyphauswahlfunktion. Ein Word Dokument legt somit bereits ohne ein Stylesheet alleine durch seine logische Struktur die anzuwendenden Formatierfunktionenen und somit die zur Formatierung zu erzeugende Hierarchie der Formatierfunktionen fest (siehe Abbildung 2.29). Als Konsequenz aus dieser automatischen Zuordnung jeweils *einer* Formatierfunktion zu einem logischen Objekt eines Dokumentes ergibt sich sofort, dass die Tiefe der hierarchischen Struktur in einem Word Dokument durch die erlaubte hierarchische Struktur von Formatierfunktionen von Word begrenzt ist [7].

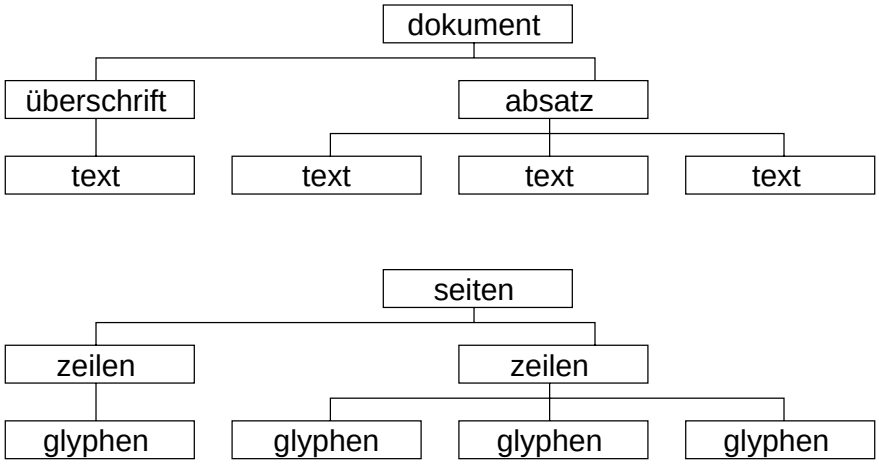

Abbildung 2.29: Korrespondierende logische Struktur und Formatierfunktionen-Hierarchie eines Dokumentes. Oben: Logische Struktur eines Word Dokumentes Unten: Hierarchie der Formatierfunktionen

In einer Style Rule für Absätze können primär Parameter für die Steuerung der auf die logischen Paragraphen anzuwendende Zeilenumbruchsfunktion angegeben werden. Mögliche Eigenschaften, die mit Hilfe von Parametern gesteuert werden können, sind unter anderem der Einzug (links, rechts, Erstzeileneinzug), der Zeilenabstand, der Abstand vor und nach einem Absatz sowie die Ausrichtung (links, rechts, zentriert, block) des Inhalts eines Absatzes. Modellhaft kann man sich vorstellen, dass die in der Style Rule spezifizierten Parameter mit Hilfe eines Containers an die Zeilenumbruchsfunktion gebunden werden, die mit dem logischen Objekt korrespondiert, für das die Style Regel definiert ist. Die Position sowohl der Formatierfunktion als auch des logischen Objektes innerhalb des gegebenen Strukturbaumes und somit der strukturelle Kontext der Anwendungsstelle spielt bei der Zuordnung der Parameter keine Rolle.

Eine Style Regel Namens B kann in Word auf eine andere Style Regel mit Namen A aufbauen. Ist in Style Regel B für einen Parameter kein expliziter Wert angegeben, so wird der Wert des Parameters aus der Style Regel A übernommen. Die Vorteile dieses Vorgehens liegen durch die eingesparte mehrfache Spezifikation eines bestimmten Wertes an mehreren unabhängigen Stellen insbesondere in der einfachen konsistenten Änderbarkeit bei von ihrer Aufgabe her voneinander abhängigen Style Regeln. Die Parameter in einer Style Regel werden also aufgrund der *statischen* Beziehung, wie Style Regeln aufeinander aufbauen, berechnet.

Analog zu dem Vorgehen bei den Style Regeln für Absätze können in Style Regeln für diejenigen Textblöcke, die logisch besonders ausgezeichnet sind, Parameter gesetzt werden, die die Formatierfunktion Glyphauswahl steuern. Dies sind unter anderem Parameter für Schriftart, -schnitt und -grad, Unterstreichung, Farbe, Hoch-/Tiefstellung, Laufweite und Darstellung des zu verarbeitenden Textes in Kapitälchen bzw. Großbuchstaben. Für nicht besonders ausgezeichnete *Standard*-Textblöcke wird ein Satz von Parametern verwendet, der in der Style Regel für den Absatz definiert ist, der den Textblock umgibt. Diese sekundäre Aufgabe der Style Regeln für Absätze erfüllt die Aufgabe eines Default-Mechanismus zur Parametersetzung für Standard-Textblöcke, wobei alle Parameter wie auch in einer normalen Style Regel für einen Textblock gesetzt werden können.

Die Erstellung von Inhaltsverzeichnissen und Indexen ist in Word ebenfalls möglich. Diese Vorgänge sind zwar mit Hilfe von durch die Benutzer frei wählbaren Parametern in gewissen Grenzen steuerbar, die angebotenen Verfahren sind aber fest im System vorgegeben und bieten keinen direkten Stylesheetmechanismus an. Vielmehr wird in [33] die Möglichkeit, wie in Word Parameter für derartige Vorgänge gesetzt werden können, als Setzen von Werten in *Property Sheets* genannt. Diese Werte werden nicht an verschiedenen Stellen im Dokument wiederverwendet, sondern werden einmalig von den Nutzern für eine spezielle Situation gesetzt.

Zusammenfassend kann man festhalten, dass im Standardbeispiel für Textverarbeitungssysteme im Bürobereich, Microsoft Word, Stylesheets nur zum Setzen von Parametern für Formatierfunktionen verwendet werden können. Die Formatierfunktionen ergeben sich bereits aus der logischen Struktur des Dokumentes. Die Parameter in den Style Regeln können nur aufgrund der statischen Vererbungshierarchie der Style Regeln vererbt werden, dynamische Beziehungen aufgrund der Anwendung einer Regel in einem Dokument haben keine Einflußmöglichkeit auf den Wert von Parametern. Auch ist es nicht möglich, mit Hilfe des angebotenen Stylesheetmechanismus die Erzeugung von z.B. Verzeichnissen zu beeinflussen.

Stylesheets in der Kombination von TeX und LaTeX

„Das wohl leistungsfähigste Formatierungsprogramm zur Erzeugung wissenschaftlich-technischer Texte stammt von Donald E. Knuth“ [36] und trägt den Namen TeX. TeX ist ein Dokumentformatierer, dem eine universelle Programmierumgebung beigestellt ist. Die enorme Leistungsfähigkeit von TeX hat jedoch ihren Preis: Die Anwendung und Ausschöpfung schon geringer Möglichkeiten zur Gestaltung eines Dokumentes setzen erhebliche Programmierkenntnisse voraus. Die Anwendung von TeX ist somit prinzipiell Programmierern vorbehalten.

Um die von TeX angebotene Leistungsfähigkeit auch Nichtprogrammierern zur Verfügung stellen zu können hat Leslie Lamport die Möglichkeit von TeX genutzt, Makros definieren zu können. In Verbindung mit dieser Möglichkeit hat er ein Programmpaket LaTeX entwickelt, das sich zwar einerseits auf TeX abstützt, andererseits aber eine *benutzerfreundliche* Schicht zwischen TeX und die Nutzer einfügt.

Die Dokumenterstellung in LaTeX erfolgt prinzipiell mit Hilfe des Paradigmas der logisch ausgezeichneten Dokumente. Das Makropaket LaTeX, das als ein Stylesheet betrachtet werden kann, übersetzt das logisch ausgezeichnete vom Benutzer erstellte Dokument in eine Eingabe, die vom Dokumentformatierer TeX verarbeitet werden kann.

Im Gegensatz zu dem Stylesheetansatz bei Word dient das Makropaket LaTeX aber nicht nur als Mechanismus zum Setzen von Parametern für Formatierfunktionen. Auch ist durch die Vorgabe der logischen Struktur im zu verarbeitenden Dokument nicht automatisch die Hierarchie der das Dokument formatierenden Formatierfunktionen festgelegt. Vielmehr kann mit Hilfe der einzelnen Style Regeln, die als Makros im Makropaket LaTeX implementiert sind, für jedes logische Objekt im Dokument vollkommen frei festgelegt werden, in welcher Form und wo es im formatierten Dokument vorkommt. So kann mit Hilfe eines LaTeX Stylesheets frei festgelegt werden, welche Dokumentbestandteile in welchen Index aufgenommen werden sollen, wie Nummerierungen vorgenommen werden sollen, u.s.w..

Die Gestaltungsfreiheiten des Stylesheetansatzes, die sich bei der Abstützung eines Systems auf die Möglichkeiten von TeX und dort frei definierbaren Makros ergeben, sind somit wesentlich umfassender als beim Stylesheetansatz von Word. In [7] wird aber ausdrücklich betont, dass die Implementierung eines Stylesheets mit Hilfe eines Makroansatzes zu schwer erstellbaren und wartbaren Systemen führt.

2.5.3 Stylesheets in Standards

CSS1

„CSS1 is a simple style sheet mechanism that allows authors and readers to attach style (e.g. fonts, colors, spacing) to HTML documents. The CSS1 language is human readable and writable, and expresses style in common desktop publishing terminology.“ [53]. CSS1 ist eine Abkürzung für Cascading Style Sheets, level1. CSS1 ist eine Entwicklung des W3C, wurde am 17.12.1996 erstmals als W3C Recommendation verabschiedet und am 11.1.1999 in überarbeiteter Form der Öffentlichkeit zur Verfügung gestellt.

Wie bereits in der Einleitung bemerkt, ist CSS1 ein Stylesheetansatz für HTML Dokumente. Die prinzipielle Idee der Erfinder von HTML war es, mit Hilfe dieser in SGML definierten logischen Auszeichnungssprache Dokumente *rein* inhaltlich auszuzeichnen und mit Hilfe von geeigneten Browsern (siehe 2.4.3) anzuzeigen. Die technische Weiterentwicklung jedoch hat HTML zu einer Sprache gemacht, mit der vom Prinzip her Hierarchien von Formatierfunktionen beschrieben werden und nicht mehr logisch ausgezeichnete Dokumente. Gründe hierfür waren insbesonders kommerzielle Anforderungen. Mit dem Siegeszug des WWW musste seine Auszeichnungssprache HTML in der Lage sein, die Formatierung von Dokumenten akribisch

bis ins letzte Detail regeln zu können, damit ein gegebenes Dokument in allen Browsern so dargestellt wird, wie es seine Ersteller wünschen.

Dieser erste Schritt der Weiterentwicklung, der HTML zu einer Beschreibungssprache von Formatierfunktion-Hierarchien gemacht hat, war den kommerziellen Anforderungen aber noch nicht vollständig gewachsen. Die Attribute für die Formatierfunktionen konnten noch nicht in geeigneter Form und ausführlich genug spezifiziert werden.

Zur Behebung dieses Mankos wurde CSS1 entworfen. Die wesentliche Aufgabe eines CSS1 Stylesheets besteht darin, die in einem HTML Dokument vorkommenden und damit gleichbedeutend die das Dokument verarbeitenden Formatierfunktionen zu attributieren. Dies geschieht mit Hilfe von Style Regeln. Jede CSS1 Style Regel besteht aus einem *Selektor* und einem *Deklarationsteil*. Der Selektor spezifiziert, auf welche Elemente respektive Formatierfunktion der Deklarationsblock der Regel angewendet werden kann. Der Deklarationsblock enthält eine Menge von Attribut-Wert-Paaren, die zur Steuerung der Formatierfunktionen verwendet werden. Ein CSS1 Stylesheet überführt also eine gegebene Hierarchie von Formatierfunktionen in eine attributierte Hierarchie von Formatierfunktionen.

Im Gegensatz zu dem Stylesheetansatz bei Word bauen Style Regeln in CSS1 nicht statisch aufeinander auf. Statt dessen können die Werte für Attribute, die in einer Regel für ein logisches Objekt gesetzt werden, an die Kinder im Strukturbaum des Dokumentes vererbt werden. Statt einer statischen Beziehung für die Werte von Attributen aufgrund der Aufschreibung der Style Regeln in einem Stylesheet besteht hier also eine *dynamische* Beziehung (siehe hierzu auch [7]) für die Attributwerte aufgrund ihrer Setzung im Strukturbaum des Dokumentes. Verdeutlicht wird dies in folgendem Beispiel:

Die Regel

```
H1 {font-size:  18pt color:  blue}
```

setzt für den Inhalt des H1 Elements das Attribut `font-size` auf den Wert `18pt` und das Attribut `color` auf den Wert `blue`. In einer Konstellation, bei der ein logisches H1 Objekt ein logisches EM Objekt enthält und für das logische EM Objekt weder für das `font-size` noch für das `color` Attribut ein Wert gesetzt wird, werden beide Attributwerte an das logische EM Objekt vererbt. Aufbauend auf der Vererbung können Werte in einer Style Regel relativ verändert weiter nach unten gegeben werden. Eine typische Anwendungsstelle für dieses Vorgehen ist das Setzen der Fontgröße in einem logischen H1 Objekt relativ zur Fontgröße der Umgebung.

Die Anwendung einer Style Regel in CSS1 kann an eine Bedingung geknüpft werden, die im Wesentlichen aufgrund der Dokumentstruktur berechnet werden kann. Dazu können aber nur die *Vorfahren* eines Elementes im Dokumentbaum herangezogen werden.

Um dieses Konzept der bedingten Regelanwendung in CSS1 zu stärken und praxisgerechter zu gestalten, kann eine Bedingung zusätzlich an der Wert von bestimmten Attributen geknüpft werden. Weiterhin können auch Bedingungen vorgegeben werden, die erst zur Laufzeit eines Formatierers berechnet werden können (z.B. die Eigenschaft, in der ersten Zeile eines Absatzes zu sein).

Der Name CSS1 rührt von der Eigenschaft her, dass ein CSS1 System eine geordnete Liste (*Kaskade*) von Style Sheets [38] als Eingabe erhält. „One of the fundamental features of CSS is

that style sheets cascade; authors can attach a preferred style sheet, while the reader may have a personal style sheet to adjust for human or technological handicaps.“ Für die Eigenschaft von CSS1, dass mehrere Stylesheets gleichzeitig die Attributierung eines Dokumentes steuern können, gibt es nach [53] zwei Hauptgründe: *Modularität* und *Interessensausgleich* zwischen den Autoren und Lesern eines Dokumentes. Zur Auflösung von Konflikten sind Style Regeln mit Prioritäten versehen, die sich nach der Herkunft der Style Regel, ihrer Position in einem Stylesheet, der Art des Selektors und noch zusätzlich spezifizierbaren Angaben berechnen. Diese Priorisierung wird *Kaskadierung* genannt.

CSS2

CSS2 [54] ist ein Stylesheetmechanismus, der auf CSS1 aufbaut. Nach Angaben in [54] sind *fast* alle gültigen CSS1 Stylesheets auch gültige CSS2 Stylesheets. Folglich unterstützt CSS2 im Wesentlichen vollständig die in CSS1 angebotenen Konzepte. CSS2 liegt aktuell als W3C Empfehlung vom 12.5.1999 vor.

Die in CSS2 eingeführte Neuerung, die CSS2 konzeptuell von CSS1 deutlich abhebt, ist die Tatsache, dass in CSS2 ein gegebenes Dokument nicht mehr automatisch die Hierarchie der Formatierfunktionen vorgibt, mit deren Hilfe es verarbeitet wird. Statt dessen ist es in CSS2 möglich, mit Hilfe der Style Regeln logischen Objekten Formatierfunktionen zuzuweisen. Im Gegensatz zu CSS1 können also mit Hilfe des CSS2 Stylesheetansatzes auch Dokumente zur Verarbeitung aufbereitet werden und in eine Formatierfunktionen-Hierarchie überführt werden, die nicht mit Hilfe einer einzigen fest vorgegebenen Auszeichnungssprache (HTML) ausgezeichnet sind.

CSS2 stellt ein konzeptuelles Verarbeitungsmodell zur Verfügung, das in eindeutiger und übersichtlicher Weise festlegt, wie ein CSS2 System ein gegebenes Dokument samt Stylesheet verarbeitet. Das Verarbeitungsmodell besteht aus mehreren Stufen und wird in Abbildung 2.30 schematisiert dargestellt.

Abschließend ist zu erwähnen, dass in CSS2 auch die Bedingungen, unter welchem Kontext eine Regel angewendet werden kann, wesentlich komplexer gestaltet werden können, als dies in CSS1 möglich ist.

DSSSL

DSSSL [30] ist eine Abkürzung für Document Style Semantics and Specification Language und wird im internationalen Standard ISO/IEC 10179 aus dem Jahr 1996 definiert. Ziel von DSSSL ist es, die strukturelle Transformation und Formatierung von SGML Dokumenten durch Zurverfügungstellung geeigneter Mechanismen systemunabhängig beschreiben zu können. „The initial focus of DSSSL is on formatting for both paper and electronic media and on the transformation of SGML documents marked up according to different DTDs.“ [30].

DSSSL ist also ein Stylesheetansatz, der für beliebige SGML Dokumente verwendet werden kann, die gemäß einer beliebigen DTD aufgebaut sind. Der Standard öffnet diese Beschränkung

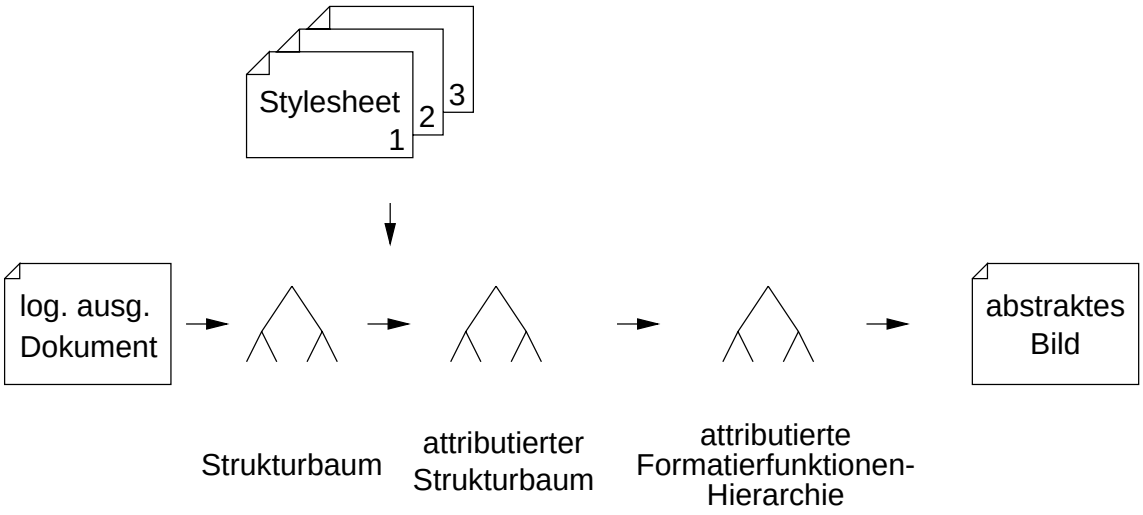

Abbildung 2.30: Das gegebene logisch ausgezeichnete Dokument wird in einen Strukturbaum überführt. Mit Hilfe der gegebenen Stylesheets wird dieser zu einem attributierten Strukturbaum angereichert. Durch Auswertung der angegebenen Formatierfunktion wird aus dem attributierten Strukturbaum eine attributierte Hierarchie von Formatierfunktionen erstellt, deren Struktur von der des atributierten Dokument Strukturbaumes aufgrund von in CSS2 möglichen Transformationen (Teile der erweiterten Aufgabe von Dokumentformatierern) in relativ engen Grenzen abweichen kann. Inhaltsverzeichnisse oder ähnliches sind aber nicht erzielbar. Die Formatierfunktionen-Hierarchie wird schließlich in die aufbereitete Ausgabe überführt.

sogar noch: „DSSSL is intended for use with documents structured as a hierarchy of elements.“ [30]. Um die Ansprüche an einen internationalen Standard zu erfüllen, werden in DSSSL nicht nur die Syntax und Semantik der Stylesheets zur Spezifikation der Verarbeitung von Dokumenten eingeführt, sondern zusätzlich auch noch Verarbeitungsmodelle. Da DSSSL zwei funktional unterschiedliche Verarbeitungsbereiche für Dokumente abdeckt, werden in DSSSL zwei angepasste Sprachen eingeführt:

Die Transformationssprache: Sie wird benutzt, um die strukturelle Transformation eines logisch ausgezeichneten Dokumentes in ein anderes Dokument zu spezifizieren. Für diese Sprache wird in DSSSL ein vollständiges Verarbeitungsmodell angegeben, das den Bereich vom Einlesen des zu verarbeitenden Dokumentes bis zur Ausgabe des transformierten Zieldokumentes beschreibt.

Die Stilsprache: Sie wird benutzt, um ein gegebenes logisch ausgezeichnetes Dokument mit Information zu versehen, damit es von einem Dokumentformatierer verarbeitet werden kann. Hierunter wird im DSSSL Standard insbesondere die Zuweisung von Formatierfunktionen und das Setzen von Attributen für diese Funktionen verstanden. Das in DSSSL definierte Verarbeitungsmodell für diese Sprache deckt, wie in der Abbildung 2.31 zu erkennen ist, nur den Teil bis zum Aufbau der Hierarchie der Formatierfunktionen ab. Der Ablauf des Formatierprozesses ist nicht mehr standardisiert.

Es ist anzumerken, dass zur Erweiterung der Ausdrucksstärke in DSSSL-Spezifikationen sowohl die Transformationssprache als auch die Stilsprache in die universelle Programmiersprache Scheme [15] eingebettet sind.

Das vollständige konzeptuelle Verarbeitungsmodell von DSSSL ist in Abbildung 2.31 dargestellt. Die in diesem Modell vorkommenden zwei Verarbeitungsprozesse Transformation und Formatierung werden mit Hilfe der Transformations- und Stilsprache gesteuert, die in Form von Stylesheets durch die Nutzer eines DSSSL Systems spezifiziert werden.

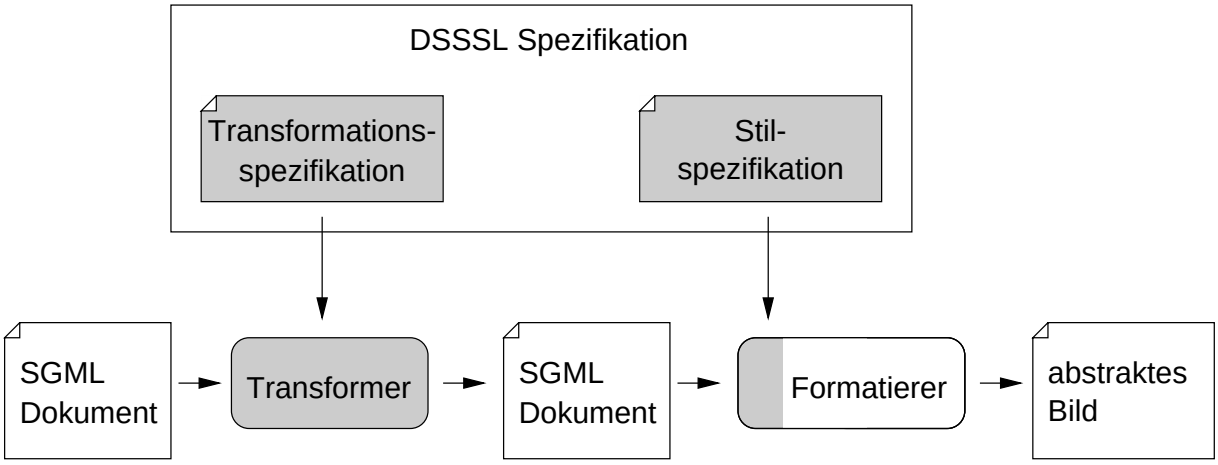

Abbildung 2.31: Das konzeptuelle Verarbeitungsmodell von DSSSL: Die grau hinterlegten Bereiche sind in DSSSL standardisiert.

Der *Transformationsprozess* überführt unter der Kontrolle einer Transformationsspezifikation ein logisch ausgezeichnetes Quelldokument in ein logisch ausgezeichnetes Zieldokument. Da die zur Steuerung dieses Vorgangs verwendete Transformationssprache in die Programmiersprache Scheme eingebettet ist, können hier mannigfaltige Transformationen ausgeführt werden. Unter anderem können hier die Aufgaben ausgeführt werden, die als die erweiterten Aufgaben von Dokumentformatierern bezeichnet wurden. Das Modell zu diesem Transformationsprozess ist in Abbildung 2.32 dargestellt.

Die Transformation wird durch sogenannte *Associations* [deu.: Verbindung] gesteuert. Eine Association ist eine Regel, die aus einer *Query* und einer *Transformation* besteht. Sie spezifiziert die Transformation von Knoten im Eingabebaum (ausgewählt über die Query) zu Knoten im Ausgabebaum. Dabei ist die Position eines Ausgabeknotens nicht fest an eine Stelle im Ausgabedokument festgelegt, sondern kann relativ zum Ergebnis einer anderen Transformation angegeben werden.

Der *Formatierprozess* des konzeptionellen Verarbeitungsmodells von DSSSL ist im DSSSL Standard nur partiell definiert. Zur Steuerung dieses Prozesses wird ein Stylesheet verwendet, das in der Stilsprache formuliert ist. Das vorrangige Ziel dieses Prozesses, soweit er in DSSSL spezifiziert ist, ist die Überführung eines gegebenen logisch ausgezeichneten Dokumentes in eine attributierte Hierarchie von Formatierfunktionen. Die Struktur des gegebenen logisch ausgezeichneten Dokumentes muss nicht zur Struktur der resultierenden Hierarchie von Formatierfunktionen äquivalent sein. Während der Erzeugung dieser Hierarchie können Anreicherungen

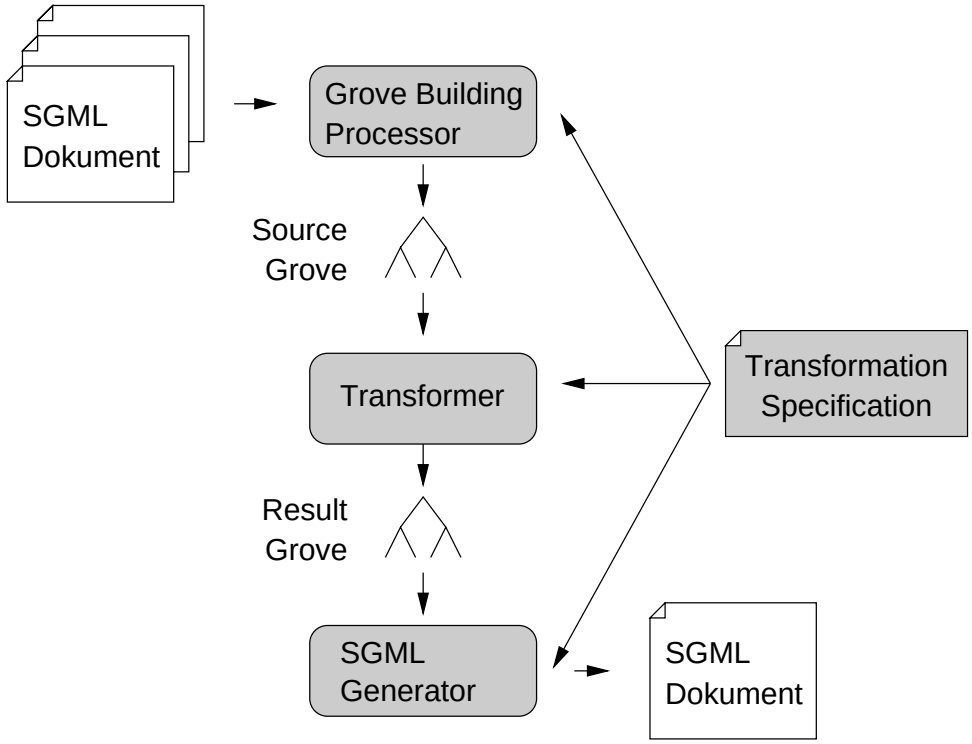

Abbildung 2.32: Der Transformationsprozess in DSSSL: Die grau hinterlegten Bereiche sind in DSSSL standardisiert. Ein oder mehrere SGML Dokumente werden mit Hilfe eines *Grove Building Processors* in einen sogenannten Source *Grove* überführt. Dieser Grove ist vom Prinzip her der Strukturbaum der Eingabedokumente, über den auch alle Attribute der logischen Objekte zugreifbar sind. Mit Hilfe eines *Transformers* wird der Source Grove unter Ausführung der in der Transformationsspezifikation gegebenen Regeln in den Result Grove überführt. Der SGML Generator linearisiert den Result Grove in ein SGML Dokument, dessen Gültigkeit gemäß einer DTD aber noch mit Hilfe eines SGML Parsers geprüft werden muss, da die Transformation ein gemäß der Ziel DTD ungültiges Dokument liefern kann.

und strukturelle Transformationen des Zielbaums vorgenommen werden. Das Modell zu dem Formatierprozess ist in Abbildung 2.33 dargestellt.

Formatierfunktionen und deren Parameter werden im DSSSL Standard zu einer Einheit zusammengefaßt, die *Flow Object* genannt wird. Zusammen definieren sie die Semantik der Verarbeitung von Dokumentinhalten: „The flow object classes and the characteristics that apply to them define the formatting appearance and behaviour of the contents of the document.“ [30]. Der DSSSL Standard legt keine Implementierungsanleitung für die Realisierung der Flow Objects fest. Es werden ausschließlich mit Hilfe eines Boxenmodells [30] die Auswirkungen beschrieben, die die Flow Objects auf ihre Inhalte haben. Im DSSSL Standard wird eine Basismenge an Flow Objects eingeführt, die in einem Stylesheet, das in der Stilsprache formuliert ist, zur Erzeugung des Flow Object Trees verwendet werden können. Die hier definierte Menge an Objekten ist am Bedarf von komplexen Bürodokumenten ausgerichtet. Mit Hilfe der einzelnen Flow Objects können komplexe Aufgaben, wie z.B. der Zeilenumbruch oder die Formatierung

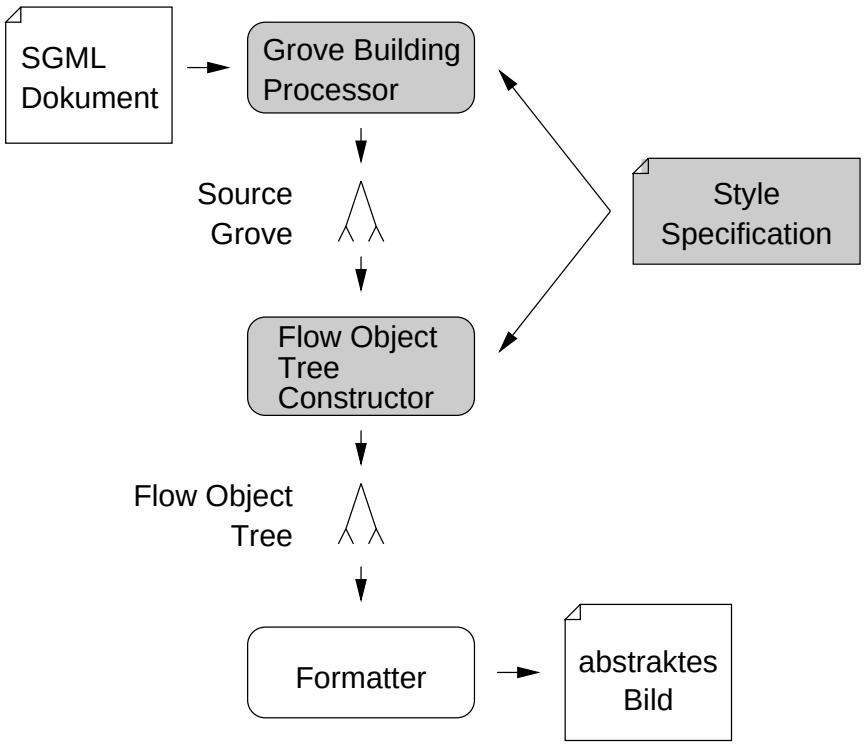

Abbildung 2.33: Der Formatierprozess in DSSSL: Die grau hinterlegten Teile sind in DSSSL standardisiert. Wie beim Transformationsprozess erstellt der Grove Building Prozessor einen Strukturbaum, von dem ausgehend mit Hilfe der Stilspezifikation der aus Flow Objecten bestehende sogenannte *Flow Objekt Tree* erstellt wird. Nicht mehr in DSSSL funktional spezifiziert ist die Arbeitsweise des Formatierers, der den Flow Objekt Tree in ein formatiertes Dokument überführt.

einer Tabelle spezifiziert werden. Die Anordnung der Flow Objects in einem Flow Object Tree muss gewissen Anforderungen genügen, um eine gültige Konfiguration darzustellen. Das prinzipielle Kriterium, das hierbei angewendet wird, ist die Eigenschaft eines Flow Objects, *inlined* oder *displayed* zu sein.

Inlined Flow Objects sind dadurch gekennzeichnet, dass sie in üblichen Dokumentformatierern von links nach rechts z.B. in Paragraphen aufeinanderfolgend angeordnet werden und zur Erstellung von Zeilen beitragen [30]. Flow Objects, die nicht inlined sind, sind displayed Flow Objects. Ihre Anordnung erfolgt im Standardfall vertikal untereinander.

Style Regeln sind in DSSSL ähnlich zu den Style Regeln in CSS aufgebaut. Logischen Objekten wird eine Formatierfunktion mit einem Satz von Werten für Attribute zugeordnet. Dies geschieht in DSSSL über die Verwendung von Flow Objects. Allerdings können in DSSSL zusätzlich bei diesen Regeln mit Hilfe von Anweisungen Anreicherungen in der erzeugten Hierarchie der Formatierfunktionen vorgenommen werden, um z.B. generierte Inhalte in das Dokument einzufügen. Aufgrund der Einbettung in eine Programmiersprache ist die Möglichkeit in DSSSL, kontextabhängige Verarbeitung durchzuführen, wesentlich umfangreicher als in CSS unterstützt.

Wie im CSS Ansatz sind in DSSSL Style Regeln nicht statisch aufeinander aufgebaut, um die Attributwerte für Formatierfunktionen zu berechnen. Vielmehr besteht auch bei diesem Ansatz eine dynamische Beziehung bei den Attributwerten aufgrund einer Vererbung im Flow Object Tree.

Mit Hilfe des DSSSL Standards kann also die beliebige strukturelle Transformation und Anreicherung eines logisch ausgezeichneten Dokumentes sowie dessen Überführung in eine Hierarchie von Formatierfunktionen beschrieben werden. Eine DSSSL Verarbeitungsanweisung, die in der Transformationssprache geschrieben ist, dient nicht der Zuweisung von Layoutinformation an logische Objekte und sollte deswegen in Anlehnung an [23] mit Specificationsheet anstelle mit Stylesheet bezeichnet werden. Die mit DSSSL Spezifikationen steuerbare Funktionalität ist insbesondere durch die Einbettung des Standards in die Scheme Programmiersprache sehr groß, gleichzeitig aber nur für Programmierer handhabbar.

Es ist an dieser Stelle anzumerken, dass gegenwärtig keine Systeme existieren, die den vollständigen DSSSL Standard implementieren. Die fortgeschrittenste Implementierung namens *Jade* stammt von James Clark und ist unter [13] zu finden.

XSL

XSL [57] ist eine Abkürzung für Extensible Stylesheet Language, ist ein Working Draft des W3C und liegt derzeit in der Version vom 16.12.1998 vor. XSL ist ein Stylesheetansatz, der zur Spezifikation der Verarbeitung von XML Dokumenten entwickelt wurde. Er stellt im Wesentlichen eine Vereinfachung des DSSSL Standards dar, um in absehbarer Zeit implementiert werden zu können.

Wie der DSSSL Standard gliedert sich XSL in zwei Teile, die Transformation von XML Dokumentstrukturbäumen und die Überführung eines Dokumentes in eine attributierte Hierarchie von Formatierfunktionen. Die verwendeten Modelle für den Transformations- als auch den Formatierprozess sind funktional analog zu den Prozessmodellen bei DSSSL aufgebaut. XSL ist zwar ebenfalls wie DSSSL in eine Programmiersprache eingebettet, diese hat aber nicht den Umfang von Scheme. Wie in DSSSL ist auch in XSL eine Menge von Flow Objects definiert, die zur Spezifikation des Aufbaus der Hierarchie von Formatierfunktionen verwendet werden kann. Die hier definierten Flow Objects sind jedoch nicht in der Komplexität steuerbar, wie dies in DSSSL möglich ist. Als Besonderheit sind auch HTML Objekte als Flow Objects in XSL zugelassen. Dies ist möglich, da wie bereits bei CSS1 gezeigt, HTML Objekte von ihrem Wesen her Formatierfunktionen sind. Durch diese Ergänzung ist es möglich, bestehende HTML-Browser als Formatierer für ein XSL System zu verwenden.

Insgesamt kann also XSL als der *kleine Bruder* von DSSSL bezeichnet werden, der in Zukunft nach Ansicht des W3C die Basistechnologie von WWW-Browsern bestimmen soll.

Kapitel 3

Softwarebasierte Unterstützung des Publikationsprozesses

3.1 Automatische Verarbeitung logisch ausgezeichneter Dokumente

Die Paradigmen im traditionellen Publikationsprozess unterliegen seit Einführung der digitalen Dokumentverarbeitung einem kontinuierlichen Wandel. Bestimmende Faktoren hierfür sind unter anderem der stetige Wunsch der Kostenminimierung, steigende Qualitätsansprüche, die Notwendigkeit kürzerer Produktionszeiten von Publikationen sowie die wachsenden Anforderungen an das Resultat von Publikationsprozessen. War früher das Ziel eines Publikationsprozesses ein gedrucktes Buch, so sind die heutigen Anforderungen im Vergleich dazu wesentlich komplexer. Oft soll als Produkt nicht nur ein gedrucktes Buch entstehen, sondern beispielsweise zusätzlich auch eine HTML- und eine CD-Version – und alle womöglich auch noch mit kundenspezifischen Anpassungen vollautomatisch erstellt.

Allgemeines Ziel im Bereich der Dokumentverarbeitung ist es deshalb, den Prozess der Dokumenterstellung vollkommen ohne einen Medienbruch mit Hilfe von digitalen Rechensystemen zu bewerkstelligen. Der gängige Begriff hierfür ist *elektronisches Publizieren*[1] (siehe hierzu auch Definition 22).

Wie in Kapitel 2 gezeigt wurde, bringt diese Entwicklung hin zum digitalen Publikationsprozess die Notwendigkeit mit sich, das Dokumentmodell der logisch ausgezeichneten Dokumente als das grundlegende Datenmodell zur Repräsentation von Dokumenten zu wählen. Das derzeit weit verbreitete Modell der hauptsächlich graphisch ausgezeichneten Dokumente bietet nicht die Möglichkeit der Erfassung all jener Informationen, die für eine automatische maschinelle Verarbeitung von Dokumenten mit den verschiedensten komplexen Aufgabenstellungen benötigt werden. Logisch ausgezeichnete Dokumente hingegen bieten durch ihre Strukturkomponente zumindest vom Konzept her die Möglichkeit, den für eine automatische Verarbeitung

[1] Der aufgrund der allgemeinen Ablösung des Wortes elektronisch konsequentere Begriff *digitales Publizieren* ist derzeit noch nicht gebräuchlich.

benötigten *Mehrwert* zur Verfügung zu stellen. Weitere erwähnenswerte Vorteile, die die Abstützung auf logisch ausgezeichnete Dokumente mit sich bringt, sind insbesondere bei der Verwendung von Normen wie SGML oder XML aktuelle Themen wie die Wiederverwendbarkeit von Dokumentbestandteilen, Archivierbarkeit, Portabilität und Recherchierbarkeit.

Einzig die fehlende Semantik (siehe 2.2.3) der logischen Objekte der logisch ausgezeichneten Dokumente bringt Probleme mit sich: Sie läßt keine allgemeine Verarbeitung beliebiger logisch ausgezeichneter Dokumente mit konventionellen Produkten zu. Eine mögliche Lösung bieten hier, wie bereits aufgezeigt, Systeme, die bestimmte Klassen von Dokumenten (mit einer a priori bekannten Menge von Tags zur Strukturierung der Dokumente) kennen und verarbeiten können.

Einen allgemeineren und somit offeneren Ansatz zur Verarbeitung beliebiger logisch ausgezeichneter Dokumente ermöglichen Systeme, die flexibel von außen steuerbar sind und so das fehlende Wissen über die Semantik der Tags ausgleichen. Im allgemeinen Fall kann diese Steuerung dokumentspezifisch, oder generischer, spezifisch für Klassen von Dokumenten (z.B. bezüglich einer DTD), mit Hilfe von *Verarbeitungsanweisungen* erfolgen.

Derartige Verarbeitungsspezifikationen können eine breite Palette von möglichen Vorgängen steuern. Einige Beispiele hierzu sind im Folgenden aufgeführt:

- **Textanalyse**: Zur Auswertung von Texten z.B. nach stilistischen Kriterien können für geeignete Systeme Verarbeitungsanweisungen erstellt werden, die das Vorkommen gewisser Wörter zählen oder die durchschnittliche Länge von Sätzen berechnen.

- **Konvertierung:** Bei Übernahme eines mit Hilfe einer Auszeichnungssprache A erfaßten Dokumentes in ein neues System, das mit der Auszeichnungssprache B arbeitet, kann es nötig sein, die im Dokument vorkommende Auszeichnung, gesteuert über eine Verarbeitungsanweisung, zu konvertieren.

- **Orthographieprüfung:** Gemäß einer Verarbeitungsanweisung kann die Orthographieprüfung eines Dokumentes aufgrund eines allgemein verfügbaren Regelwerkes oder aufgrund von lokal vorgegebenen Regeln erfolgen. Speziell bei mehrsprachigen Texten kann an gewissen Stellen eine Umschaltung nötig sein.

- **Strukturelle Transformationen und Anreicherungen:** Die Aufsplittung eines Dokumentes in mehrere Teildokumente bzw. das Zusammenfassen mehrerer kleiner Dokumente zu einem grösseren, sowie die Anreicherung eines Dokumentes um z.B. Annotationen zu gewissen logischen Objekten aus einem Reviewprozess, kann über Verarbeitungsanweisungen gesteuert werden.

- **Aufbereitung:** Zur Verarbeitung durch beliebige Programme kann es nötig sein, ein logisch ausgezeichnetes Dokument zunächst mit Hilfe einer Verarbeitungsanweisung in ein geeignetes Eingabeformat aufzubereiten.

Wenn die Verarbeitung, die mit Hilfe einer Verarbeitungsanweisung gesteuert wird, hauptsächlich der Publikation eines Dokumentes dient, nennen wir diese Verarbeitungsanweisung eine Designspezifikation.

Definition 27 (Designspezifikation) *Eine Designspezifikation ist eine Verarbeitungsanweisung für ein geeignetes System, das der Aufbereitung eines Dokumentes zur dessen Publikation dient. Mit Hilfe von Designspezifikationen werden insbesondere die Kern- und erweiterten Aufgaben von Dokumentformatierern, sowie ihnen vor- und nachgeschaltete Programme, aufgrund eines gegebenen Dokumentes und weiteren Faktoren (wie z.B. den Nutzern) gesteuert.*

3.2 Erstellungsszenarien für Designspezifikationen

Designspezifikationen im Sinne von Definition 27 sind in Verlagen bzw. im klassischen Publikationsprozess schon seit langem in Gebrauch. Aus diesem Grund betrachten wir zunächst, wie traditionell in Verlagen bei der Erstellung einer Publikation gearbeitet wird, bevor wir auf die Szenarien eingehen, wie sie bei der Arbeit mit Microsoft Word und DSSSL entstehen.

3.2.1 Designspezifikation in Verlagen

Klassischer Arbeitsablauf einer Publikation

Nachfolgend werden vereinfacht dargestellt die Arbeitsschritte skizziert, wie sie beim klassischen Arbeitsablauf einer Publikation in einem Verlag anfallen, dessen Produkt üblicherweise ein Buch ist (siehe auch [26], [47], [51], [37], [8]):

1. **Randbedingungen feststellen:**
   In gewissen Situationen sind bestimmte Teile einer Publikation bereits vor ihrem Beginn durch Randbedingungen wie ähnliche Publikationen (z.B. in derselben Buchreihe), Verlagskonventionen oder andere Vorschriften vorgegeben. Diese Randbedingungen gilt es von vornherein zu erkennen und zu berücksichtigen.

2. **Anweisungen erstellen:**
   In diesem zweiten Schritt werden alle Vorschriften zur inhaltlichen Erstellung und weiteren Verarbeitung einer Publikation bis hin zum fertigen Produkt festgelegt, wobei insbesondere die Ergebnisse aus Punkt 1 berücksichtigt werden. Der für diese Arbeit interessanteste Punkt dieses Arbeitsschrittes ist die Erstellung der sogenannten *Satzanweisung*.

   Die Satzanweisung definiert den Satz eines Manuskriptes, „d.h. Größe und Position des Satzspiegels, Spaltenbreite, Schriftgrad, Durchschuß und Schriftgrad der Werk- und Konsultationsschrift, sowie der Überschriften. Auch spezielle Schreibweisen und Trennregeln werden darin vorgegeben.“ [26] Die Satzanweisung ist somit ein Teil der Gestaltungsvorschriften für ein Dokument und wird in der Regel von speziell ausgebildeten Personen erstellt. Ziel bei der Erstellung einer Satzanweisung ist insbesonders deren typographisch gelungene Gestaltung, die wesentlich zur Qualität einer Publikation beiträgt.

   Mit der Satzanweisung wird also im Wesentlichen die Kernfunktionalität von Dokumentformatierern gesteuert. Zumindest die Vorgabe der speziellen Schreibweisen jedoch geht über die Kernfunktionalität der Dokumentformatierer hinaus.

Weiterhin wird von der Redaktion der Publikation festgelegt, welche Verzeichnisse und weiteren Teile für die Publikation aufgrund des eigentlichen Dokumentteils erstellt werden müssen.

3. **Erstellen des Manuskripts:**
   Im traditionellen Publikationsprozess dient das von einem Autor erstellte Manuskript lediglich als eine inhaltliche Vorlage für die weitere Verarbeitung in einem Verlag. Folglich ist es nahezu gleichgültig, in welcher Form das Manuskript einem Verlag zur Verfügung gestellt wird. Mögliche Formen reichen von handschriftlich erstellten Manuskripten über mit der Schreibmaschine erstellte Manuskripte bis hin zu direkt mit einem digitalen Rechensystem erstellten Vorlagen. Die Verwendung eines digitalen Systems zur Erstellung des Manuskripts dient dabei lediglich der technisch angemessenen Unterstützung des Autors bei der Erstellung des Manuskripts. Im folgenden Publikationsprozess wird das Manuskript **bis heute** typischerweise in nichtelektronischer, ausgedruckter From weiterverarbeitet.

   Die logische Struktur der Manuskripte ist in der Regel nur durch graphische Gestaltung implizit angedeutet und nicht explizit vermerkt.

4. **Korrektur des Manuskripts:**
   Nach Übergabe des Manuskripts an den Verlag wird es dort inhaltlich und stilistisch aufgrund der bereits erstellten Vorgaben auf deren Einhaltung geprüft. Dieser Schritt trägt wesentlich zur Erreichung eines angezielten Qualitätsniveaus einer Publikation bei. Er wird in der Regel nicht auf digitalem Material durchgeführt, sondern vielmehr konventionell als handschriftliche Anmerkungen auf einem Papierausdruck.

5. **Auszeichnen, Erfassen und Setzen des Manuskripts:**
   In diesem Schritt wird die durch einen Designer erstellte Satzanweisung durch einen Lektor auf das vom Autor gelieferte und bereits korrigierte Manuskript angewendet. Zwischenergebnis dieser Phase ist ein traditionell ausgezeichnetes Dokument, das eine genaue Anweisung für einen Setzer ist, wie er ein Dokument mit Hilfe der ihm technisch zur Verfügung stehenden Hilfsmittel zu setzen hat. Ergebnis dieser Arbeit des Setzers und zugleich Endergebnis dieses Arbeitsschrittes sind die *Satzfahnen*, die als Vorlage zum Druck verwendet werden. Das Ergebnis dieser Phase wird vom Autor der Publikation in der Regel noch einmal korrekturgelesen.

6. **Erstellen der Titelei und des Anhangs:**
   Erst nach der Erstellung des gesetzten Manuskripts können davon abhängige Dokumentbestandteile wie Inhaltsverzeichnisse und Stichwortverzeichnisse erstellt werden.

   Die Erstellung dieser Teile ist allerdings ein redaktioneller Prozess, der nicht in der Satzanweisung für eine Publikation festgelegt ist. Die Vorschriften, die diesen redaktionellen Prozess regeln, sind aber ebenfalls der Designspezifikation für den Publikationsprozess zuzurechnen. Auch die in dieser Phase erstellten Dokumentbestandteile müssen von einem Setzer gesetzt werden. Zusammen mit dem gesetzten Manuskript ergeben sie die Vorlage für den nachfolgenden Realisierungsprozess.

7. **Weiterer Ablauf:**
   Mit Fertigstellung der Titelei und des Anhangs kann der eigentliche Druckprozess und die Weiterverarbeitung bis hin zum fertigen Produkt, einem Buch, beginnen. Dazu wird bei größeren Auflagen üblicherweise ein Film als Vorlage für den Druckprozess erstellt, bei kleineren Auflagen kann der Druck aber auch direkt mit Hilfe von geeigneten Laserdruckern erfolgen. Im Anschluss an den Druck können die einzelnen Seiten gefalzt, geschnitten und anschließend zu einem Buch gebunden werden.

Durch eine Designspezifikation steuerbare Prozesse

Bei der Analyse des Arbeitsablaufs einer Publikation in einem Verlag hat sich gezeigt, dass verschiedene Arbeitsbereiche mit Hilfe einer Designspezifikation gesteuert werden können:

1. **Inhaltliche und stilistische Korrektur:**
   Der erste im Rahmen einer Publikation steuerbare Prozess ist die inhaltliche und stilistische Prüfung eines Manuskripts auf Übereinstimmung mit den in der Satzanweisung getroffenen Vorgaben. Hierbei wichtige Faktoren sind z.B. die Stimmigkeit mit anderen Publikationen aus einer Publikationsreihe, sowie die Überprüfung der Orthographie des Manuskripts gemäß getroffenen Vorgaben.

2. **Auszeichnen des Manuskripts:**
   Der zweite steuerbare Prozess ist die graphische Auszeichnung des Manuskripts durch einen Lektor. Die Vorgaben hierzu sind, wie schon beim ersten Schritt, in der Satzanweisung festgelegt.

   Typische Vorgaben, die die Satzanweisung für diesen Arbeitsschritt bereithält, sind Regeln über die Art der Ziffern, die für die verschiedenen Nummerierungen in einem Dokument verwendet werden sollen, und die Art der zu verwendenden Anführungszeichen. In typographisch hochwertigen Publikationen ist es üblich, je nach strukturellem Kontext verschiedene Anführungszeichen zu verwenden. Ein Beispiel wäre die Verwendung der Anführungszeichen ›...‹ für Begriffe und »...« für Zitate. Auch die Zuordnung der verschiedenen Schriften zu den einzelnen logischen Objekten eines Dokumentes sind hier, geknüpft an Bedingungen, geregelt. So werden Überschriften oft in einer anderen Schrift gesetzt, als der normale Fließtext.

   Bei den Vorgaben, die für den Prozess der Auszeichnung eines Manuskripts in der Satzanweisung gemacht werden, handelt es sich also durchwegs um Werte, die aufgrund statischer Informationen über ein Manuskript auf dieses angewendet werden können.

3. **Setzen des Manuskripts:**
   Auch der dritte steuerbare Prozess wird von der Satzanweisung geleitet: Die Umsetzung des graphisch ausgezeichneten Manuskripts in die Satzfahnen (der eigentliche Formatierprozess).

In einer Satzanweisung sind zur Steuerung dieses Prozesses unter anderem Vorgaben zur Ausführung der Trennung von Wörtern und der Realisierung von lebenden Kolumnentiteln enthalten. Auch für den Satz des Manuskripts enthält die Satzanweisung kontextabhängige Vorgaben. Typisches Beispiel sind hier Vorgaben für die Trennung von Wörtern: Üblicherweise wird verlangt, dass Vorsilben minimal 2 Zeichen lang sein müssen, Nachsilben minimal 3 Zeichen und zusätzlich, dass nicht mehr als 2 aufeinanderfolgende Zeilen getrennt werden dürfen.

Im Gegensatz zu Punkt zwei, dem Auszeichnen des Manuskripts, werden hier also die Anweisungen in einer Satzanweisung umgesetzt, die dynamische Aspekte der Verarbeitung eines Dokumentes während des Formatierprozesses betreffen.

Zusammengenommen sind die Punkte zwei und drei, unter Vernachlässigung von Qualitätsaspekten, äquivalent zu der Steuerung der Kernaufgaben von Dokumentformatierern mit Hilfe von Stylesheets.

4. **Erstellen der Titelei und des Anhangs:**
   Der vierte von der Designspezifikation steuerbare Prozess ist die Erstellung der Titelei und des Anhangs, nachdem der eigentliche Textteil des Dokumentes gesetzt wurde. Die Titelei umfaßt Punkte wie das Impressum, Inhaltsverzeichnis, Abbildungs-, Tabellen-, und Formelverzeichnis, Widmung, Geleitwort und Vorwort. Der Anhang kann gesammelte Anmerkungen, Literaturverzeichnis, Glossar, Register, einen Nachtrag sowie ein Schlußwort umfassen.

5. **Technische Realisierung:**
   Zuletzt kann mit Hilfe der Designspezifikation auch noch der eigentliche Produktionsprozess einer Publikation gesteuert werden.

Bewertung der klassischen Designspezifikation in Verlagen

Ein offensichtlicher, obgleich wichtiger Punkt, ist die Tatsache, dass mit Hilfe der in Verlagen erstellten Designspezifikationen Publikationen in der professionellen Qualität erstellt werden können, wie wir es aufgrund täglicher Erfahrung im Umgang mit verfügbarer Literaur gewöhnt sind. Folglich ist der Umfang der Prozesskette, den derart in Verlagen erstellte Designspezifikationen steuern können, *vollständig*, und deckt den gesamten Publikationsablauf für das Produkt *gedrucktes Buch* ab.

Die steuerbaren Prozesse haben sich aufgrund von Erfahrungen, Überlieferungen und Traditionen entwickelt, sind komplex und verfügen über einen großen Satz von Parametern, die ihre Abläufe beeinflussen können. Die Designspezifikationen, die zur Steuerung dieser Prozesse erstellt werden, sind somit präzise technische Vorgaben. Sie werden aber nicht mit Hilfe von exakten, syntaktisch formal genau definierten Mechanismen oder gar in Form von Programmen, die als Eingabe für Rechensysteme dienen können, formuliert. Vielmehr werden sie oft handschriftlich und mit Hilfe von Formularen erstellt. Ihre Auswertung und Umsetzung erfolgt durch Menschen, die mit ihrer Hilfe, evtl. unter Einbeziehung geeigneter Maschinen, die nötigen Prozesse abarbeiten.

Zur Erstellung von Designspezifikationen in Verlagen wird also eine zielgerichtete, aufgabenorientierte Vorgehensweise angewendet, die es den Erstellern von Designspezifikationen möglichst einfach macht, ihre Aufgaben zu erfüllen, ohne sie mit Formalismen zu belasten, die ihre Aufgabe unnötig erschweren. Erreicht wird diese optimale Arbeitsbedingung durch die strikte Arbeitsteilung und den hohen Personaleinsatz des klassischen Publikationsablaufs in einem Verlag.

Technische Weiterentwicklung von Designspezifikationen

Es ist zu beachten, dass die bisher angewandte Vorgehensweise bei Publikationen zwar qualitativ optimale Ergebnisse erzielen kann, aufgrund des geringen Einsatzes von heute technisch gegebenen Möglichkeiten allerdings nicht in allen Fällen bezüglich weiteren Bewertungskriterien optimale Ergebnisse erzielt.

Ein Beispiel ist die Produktion von Tagungsbänden (siehe [8]), die die Ergebnisse laufender Forschung enthalten und deshalb möglichst zügig erscheinen sollen. Allein schon diese zügige Produktion ist schwer im Rahmen einer klassischen Verlagsproduktion zu erzielen. Hinzu kommt aber noch, dass der Leserkreis aufgrund der sehr speziellen Inhalte hochgradig spezialisiert ist und dementsprechend klein ausfällt. Als Konsequenz daraus lassen sich nur sehr kleine Auflagen verkaufen, wodurch aufgrund des hohen Produktionsaufwands hohe Kosten pro verkauftem Buch entstehen.

Aufgrund dieser beiden Unzulänglichkeiten hat man den Publikationsprozess für derartige Publikationen in den Verlagen geändert und produziert Tagungsbände auf Grundlage von sogenannten *camera ready copies*. D.h., im Wesentlichen wird nur noch die technische Realisierung der Publikation im Verlag wahrgenommen, für alle anderen, bisher von Verlagen ausgeübten Arbeitsschritte, sind die Autoren selbst verantwortlich.

Der stark arbeitsteilige Prozess, der in den Verlagen von den jeweiligen Spezialisten ausgeführt wurde und deshalb Produkte hoher Qualität hervorbringen konnte, liegt bei diesem geänderten Vorgehen in der alleinigen Verantwortung des Autors. Neben der eigentlichen Erstellung der Inhalte einer Publikation wird er in diesem Fall auch mit für ihn fachfremden Aufgaben belastet, für die er wahrscheinlich nicht qualifiziert ist. Eine Tatsache, unter der auf jeden Fall die Qualität der Publikation leiden wird.

Um diesem Qualitätsverlust entgegenzuwirken, versuchen Verlage ihren Publikationsprozess umzustellen. Technischer Kernpunkt dieser Umstellung ist die Übernahme des von Autoren gelieferten Manuskripts in geeigneter digitaler Form und dessen Weiterverarbeitung mit Hilfe von geeigneten digitalen Rechensystemen ohne einen Medienbruch.

Aus diesem Umstieg auf ein digitales Publikationsverfahren ergibt sich unmittelbar ein zweiter Vorteil: Dieser nun digitale Publikationsprozess kann bei der Realisierung bisher nur aufwendig durchführbarer Projekte wie der gleichzeitigen Produktion eines Buches in gedruckter Form und zusätzlich z.B. einer HTML-Variante direkte Weiterverwendung erfahren. Das hierbei verwendete Schlagwort ist *Cross-Media Publishing*. Eng damit verbunden ist auch das *User Customized Publishing*, dessen Ziel die am individuellen Konsumenten angepasste automatische Erstellung eines Publikationsproduktes ist.

Als Konsequenz dieser Abstützung des Publikationsprozesses auf digitale Rechensysteme wird es allerdings nötig, auch die zukünftig zu erstellenden Designspezifikationen in für die verwendeten Systeme geeigneter digitaler und somit maschinenverarbeitbarer Form zur Verfügung zu stellen. Oberstes Ziel sollte es dabei sein, neben der Erhaltung der gewohnten Produktionsqualität auch den Erstellern der Designspezifikationen ihre Arbeit zumindest so einfach wie bisher zu gestalten, damit sie sich weiterhin auf ihre wesentliche Arbeit konzentrieren können, ohne von fachfremden Aufgaben belastet zu werden.

3.2.2 Designspezifikation in Microsoft Word

Arbeitsablauf einer Publikation

Der Arbeitsablauf einer Publikation mit Hilfe von Microsoft Word hat in der Regel die Erstellung eines auf Papier gedruckten *buchähnlichen* Produktes als Ziel. Typische Anwendungsgebiete sind vor allem die Erstellung von Bürodokumenten, Studienarbeiten und privaten Schriftsätzen. Ein weiteres typisches Einsatzszenario für Microsoft Word ist die Erstellung von camera ready copies für die Weiterverarbeitung in einem Verlag. Die Qualität der erstellbaren Produkte ist üblicherweise geringer als die von in Verlagen erstellten Publikationen. Im Folgenden wird die typische Erstellung einer Studienarbeit mit Hilfe von Microsoft Word bertrachtet.

Prinzipiell liegt auf der Hand, dass auch bei diesem Arbeitsablauf dieselben Schritte wie bei einer klassischen Verlagspublikation durchlaufen werden müssen. In diesem Fall liegt aber die Ausführung aller Schritte in der Hand des Autors der Publikation. Für die Steuerung existiert in aller Regel keine Designspezifikation zu Beginn der Publikation. In Ausnahmefällen werden jedoch Vorgaben von den die Arbeit betreuenden Lehrstühlen bzw. des Prüfungsamtes bezüglich der zu erstellenden Titelei und des Anhangs, sowie stilistische Vorgaben („wenn möglich, Verwendung deutscher Begriffe“) getroffen.

Ansonsten ist es die Aufgabe des Autors, vor und während der Publikation sich eine eigene Designspezifikation zu erarbeiten. Die dabei immer gegebene Gefahr liegt in der fehlenden Ausbildung normaler Autoren bezüglich der typographisch guten Gestaltung von Dokumenten. Dies resultiert oft in vollkommen unzulänglichen Designspezifikationen (insbesondere Satzanweisungen), in denen alle typographisch machbaren Konstrukte auf jeder Seite eingesetzt werden, bzw. im anderen Extrem, viel zu sparsam oder garnicht eingesetzt werden.

In einem listenartigen Überblick ergibt sich für den Publikationsablauf folgende Struktur:

1. **Vorgaben des Lehrstuhls und Prüfungsamtes einholen:**
   Aufgrund bereits bestehender Arbeiten und Vorgaben ergeben sich Vorschriften für die Erstellung der Publikation, die beachtet werden müssen.

2. **Designspezifikation erstellen:**
   Mit den Vorgaben und aufgrund eigener Vorstellungen wird mit Hilfe der in Microsoft Word (siehe 2.5.2) gegebenen Möglichkeiten ein Stylesheet erstellt, das Teile einer in Verlagen angewandten Satzanweisung realisiert. Implizit ergeben sich durch das Stylesheet die zur logischen Auszeichnung verwendbaren Tags. Dieses Stylesheet kann später automatisch auf das erstellte Manuskript angewendet werden.

3. **Manuskript erstellen und korrigieren / Ergänzen der Designspezifiktion / Erstellen Titelei und Anhang:**
   In einem eng miteinander verzahnten Prozess erstellt der Autor inkrementell das Manuskript, ergänzt nötigenfalls die Designspezifikation und läßt aufgrund der im System fest vorgegebenen Möglichkeiten Komponenten der Titelei und des Anhangs erstellen.

4. **Weiterer Ablauf:**
   Mit Abschluss des dritten Schrittes kann das ausgedruckte Manuskript oder eine andere geeignete Vorlage (evtl. eine PS-Datei) an einen Copy-shop oder einen Verlag zur technischen Fertigstellung übergeben werden.

Durch die Designspezifikation steuerbare Prozesse

In den bisherigen Betrachtungen dieser Arbeit hat sich gezeigt, dass sich mit Hilfe der in Microsoft Word in Form von Stylesheets erstellbaren Designspezifikationen weniger Prozesse und Funktionen steuern lassen, als dies bei Designspezifikationen im klassischen Publikationsprozess möglich ist. Vorweggreifend kann man festhalten, dass die steuerbaren Bereiche lediglich Teile der klassischen Satzanweisungen sind.

1. **Orthographische Prüfung:**
   Die Orthographie in einem erstellten Dokument kann aufgrund wählbarer vorgegebener und frei definierbaren Wörterbüchern geprüft werden. Stilistische Korrekturen sind nicht möglich.

2. **Teilweises Auszeichnen des Manuskripts:**
   Mit Hilfe der erstellbaren Stylesheets ist es nur möglich, Teile des Auszeichnungsprozesses, der ein (partiell) logisch ausgzeichnetes Dokument in ein vollkommen graphisch ausgezeichnetes Dokument überführt, von Microsoft Word ausführen zu lassen. Andere Teilbereiche müssen aufgrund der prinzipiell in Microsoft Word verwendeten Paradigmen (nicht alles in einem Dokument kann logisch ausgezeichnet werden) von Hand durch den Autor graphisch ausgezeichnet werden.

3. **Teilweises Setzen des Manuskripts:**
   Wie schon beim Auszeichnen können auch für das Setzen des Manuskripts in Microsoft Word nicht alle nötigen Parameter spezifiziert werden, damit dieser Prozess automatisch durchgeführt werden kann. Insbesondere können hier nur wenige Parameter gesetzt werden, die für komplex zu erzielende typographische Anforderungen nötig sind.

Bewertung der Designspezifikation in Microsoft Word

Der funktionale Umfang, der mit Hilfe von Designspezifikationen in Microsoft Word gesteuert werden kann, beschränkt sich im Prinzip auf die Kernaufgaben von Dokumentformatierern. Die Steuerung der Erstellung von Titelei und Anhang ist nur in engen Grenzen aus vorgegebenen Listen *wählbar*, besondere Vorgaben können nicht gemacht werden. Auch ist die mögliche

Komplexität der Designspezifikationen bei weitem nicht so umfassend, wie sie beim Vorgehen in Verlagen möglich ist. Die Erstellung der Designspezifikationen erfolgt in für graphische Benutzungsoberflächen üblichen Dialogfenstern, wobei die erstellbaren Regeln statisch aufeinander aufbauen (siehe 2.5.2) können. Strukturelle Abhängigkeiten können in den Stylesheets nicht berücksichtigt werden und müssen deshalb vom Autor von Hand realisiert werden.

3.2.3 Designspezifikation in DSSSL

Arbeitsablauf einer Publikation

Der DSSSL Standard ist mit dem Ziel konzipiert worden, den im professionellen Verlagswesen auftretenden Anforderungen in Bezug auf die Erstellung von Designspezifikationen entsprechen zu können. Kernbereich der Prozesse, die mit in DSSSL erstellten Designspezifikationen gesteuert werden sollen, sind die Aufbereitung logisch ausgezeichneter Dokumente für den Publikationsprozess sowie die Steuerung der Kern- und der erweiterten Aufgaben von Dokumentformatierern.

Aufgrund dieser Zielrichtung ist es somit nicht möglich, mit DSSSL Designspezifikationen den Publikationsprozess in der umfassenden Weise zu steuern, wie dies bei dem Vorgehen in einem Verlag möglich ist. Zudem ist es nötig, bei einer durch eine DSSSL Designspezifikation gesteuerte Publikation eine logische Auszeichnungssprache zur grundlegenden Manuskripterstellung zu definieren.

Der in einem auf DSSSL gestütztes System ablaufende Publikationsprozess kann prototypisch wie folgt aussehen:

1. **Randbedingungen feststellen:**
   Wie auch schon bei den anderen beschriebenen Publikationsabläufen gilt es, vor Beginn der eigentlichen Publikation dessen Randbedingungen zu evaluieren und zu berücksichtigen.

2. **Anweisungen erstellen:**
   In diesem Schritt ist zunächst die logische Auszeichnungssprache für die Publikation auszuwählen. Im Gegensatz zum klassischen Publikationsverfahren wird bei einer DSSSL gestützten Publikation das zu verarbeitende Dokument in digitaler, logisch ausgezeichneter Form zur digitalen Weiterverarbeitung angeliefert.

   Im Anschluss an diese Auswahl der logischen Auszeichnungssprache wird mit Hilfe der in DSSSL definierten Transformations- und Stilsprache die Designspezifikation erstellt, die nicht nur Teile einer klassischen Satzanweisung realisiert, sondern auch noch zusätzlich Teile der redaktionellen Verarbeitung regelt. Sowohl die Erstellung der Auszeichnungssprache als auch die Erstellung der Designspezifikation geschieht mit Hilfe programmiersprachlicher Konstrukte und ist nur von speziell ausgebildeten Personen durchführbar.

3. **Erstellen des Manuskripts / Korrektur:**
   Das Manuskript muss vom Autor in digitaler Form mit Hilfe der gegebenen logischen Auszeichnungssprache erstellt werden. Die Korrektur des Manuskripts geschieht in digitaler Form und liefert als Ergebnis wieder ein digitales logisch ausgezeichnetes Dokument. Die Korrekturen erfolgen nicht unbedingt vollständig durch die Designspezifikation gesteuert statt. Gewisse Prozesse (z.B. die stilistische Korrektur) werden von Menschen aufgrund von nicht in DSSSL formulierten Spezifikationen ausgeführt.

4. **Erstellen Titelei / Auszeichnen und Setzen des Manuskripts:**
   In diesem Schritt wird maschinell die Designspezifikation auf das erstellte Manuskript angewendet. Das Zwischenergebnis dieser Phase ist im Prinzip ein traditionell ausgezeichnetes Dokument, das bereits die Titelei und den Anhang beinhaltet. Dieses Zwischenergebnis muss von einem geeigneten Formatierer umgesetzt werden.

5. **Weiterer Ablauf:** Je nach Produkt der vorherigen Phase kann dieses nun in weiteren Schritten weiterverarbeitet werden, die sich der Steuerung einer DSSSL Designspezifikation jedoch entziehen.

Durch eine DSSSL Designspezifikation steuerbare Prozesse

Der Umfang der in einem DSSSL System steuerbaren Prozesse ist geringer als beim Publikationsprozess in einem Verlag. Im Wesentlichen können die Kern- und die erweiterten Funktionen von Dokumentformatierern gesteuert werden.

1. **Orthographische Prüfung:**
   Vergleichbar zu den Möglichkeiten bei Microsoft Word ist es möglich, eine Orthographieprüfung anzustoßen.

2. **Erstellen von Titelei und Anhang:**
   Der erste wesentliche durch eine DSSSL Spezifikation gesteuerte Teil ist die Erstellung von Titelei und Anhang einer Publikation. Zeitlich kann die Ausführung dieses Punktes mit dem automatischen graphischen Auszeichnen des Manuskriptes zusammenfallen.

3. **Auszeichnen des Manuskripts:**
   Die Überführung des in digitaler Form vorliegenden logisch ausgezeichneten Manuskriptes in die digitale graphisch ausgezeichnete Form kann ebenfalls mit Hilfe einer DSSSL Spezifikation gesteuert werden, wobei komplexe Bedingungen spezifiziert sein können.

Bewertung der Designspezifikationen in DSSSL

Designspezifikationen in DSSSL sind nicht in der Lage, den vollständigen in Verlagen durchgeführten Publikationsprozess zu steuern. Sie können aber prinzipiell alle Kern- und erweiterten Aufgaben von Dokumentformatierern ansteuern und ermöglichen damit die Steuerung aller Komponenten in der Druckvorstufe, die nach dem momentanen technischen Stand maschinell

durchführbar sind. Die technische Zielrichtung von DSSSL ist es, mehr als nur die Erstellung des Produktes *gedrucktes Buch* zu unterstützen. Somit sind die technischen Möglichkeiten im Vergleich zum Vorgehen bei Verlagen ausgeweitet worden.

Problematisch gestaltet sich jedoch die Erstellung der Designspezifikationen in DSSSL. Im Gegensatz zum klassischen Vorgehen, bei dem die Designspezifikation handschriftlich in einer den Designern vertrauten Notation vorgenommen wurde, ist es bei der Spezifikation mit DSSSL nötig, die Designspezifikation zu *programmieren*. Dies ist ein sehr komplexer Vorgang, der ein sehr großes Maß an Programmierkenntnissen erfordert. Es ist nur schwer vorstellbar, dass die bisherigen Ersteller von Designspezifikationen diese Kenntnisse haben. Eine Befragung von technischen Fachleuten [24, 5] untermauert diese Aussage.

3.3 Analyse der Erstellungsszenarien

Bei der Diskussion der Erstellungsszenarien für Designspezifikationen hat sich gezeigt, dass mit Hilfe der in Verlagen erstellten Designspezifikationen ein typischer Publikationsprozess über alle Phasen hinweg gesteuert werden kann. Die dabei erstellbaren Produkte haben einen qualitativ hohen Standard, der durch optimale Arbeitsteilung, gute Anpassung an die Fähigkeiten des Personals und einen leider hohen Personaleinsatz erreicht wird.

Es zeigt sich also neben dem hohen erreichbaren Qualitätsniveau, dass der in den Verlagen betriebene Publikationsprozess zu langsam ist und zu hohe Kosten verursacht. Insbesondere ist diese Problematik auf die mangelnde Automatisierung durch zu geringe Abstützung auf digitale Systeme zurückzuführen.

Weiterhin fällt auf, dass das übliche Ziel eines Publikationsprozesses in einem Verlag *ein gedrucktes Buch* ist. Aufgrund dieser Tatsache kann man den klassischen Publikationsprozess in Verlagen auch mit den Begriffen *Single Product Publishing* und *Single Media Publishing* bezeichnen. Heutige Marktanforderungen gehen aber über diese Möglichkeiten hinaus. Gefordert wird heute ein Publikationsprozess, der möglichst schnell und kostengünstig mehrere Produkte in mehreren Zielmedien erstellen kann und das wenn möglich automatisch ohne ein Zutun von Menschen und noch dazu angepasst an einen individuellen Kunden. Die Schlagworte hierfür sind *Multiple Product Publishing*, *Cross-Media Publishing* sowie *User Customized Publishing*.

Typische Dokumentverarbeitungssysteme wir Microsoft Word sind technisch nicht in der Lage, die vom Prinzip her auf jeden Fall maschinell durchführbaren Publikationsprozessschritte zu übernehmen und auszuführen. Dies betrifft ausdrücklich die Steuerung der Kern- und der erweiterten Aufgaben von Dokumentformatierern und dies sowohl vom funktionalen Umfang her, als auch in Bezug auf die erzielbare Qualität. Ein weiterer Punkt ist, dass auch hier nur das Single Product Publishing unterstützt wird.

Im Gegensatz zu den typischen Dokumentverarbeitungssystemen wie Microsoft Word ist der DSSSL Ansatz grundsätzlich technisch in der Lage, die im Publikationsprozess automatisierbaren Prozesse zu unterstützen. In Zusammenhang mit der Verwendung logisch ausgezeichneter Dokumente ist es auch prinzipiell möglich, Multiple Product Publishing zu betreiben, wenngleich bisher praxisrelevante Ergebnisse noch auf sich warten lassen – eine vollständige

Implementierung des DSSSL Standards ist derzeit nicht verfügbar und wird in absehbarer Zeit auch nicht verfügbar sein, ebenso wie ein DSSSL fähiger Dokumentformatierer.

Auch ist der Designspezifikationsprozess in DSSSL in keinerlei Weise an die Bedürfnisse der bisherigen Ersteller von Designspezifikationen im Publikationsprozess der Verlage angepasst. Konnten sich die Ersteller bisher in einer an sie angepassten Notation ausdrücken, so ist es in DSSSL nötig, die gesamte Designspezifikation mit Hilfe einer Programmiersprache zu *programmieren*, anstatt sie zu *spezifizieren*. Durch die Wahl der Programmiersprache Scheme (einem LISP Dialekt), die eine Präfixnotation und weitere ungeübten Personen nur schwer zugängliche Konzepte und Schreibweisen verwendet, wird dieser Vorgang noch einmal zusätzlich verkompliziert.

3.4 Aufgabenstellung

Der klassische Publikationsprozess in Verlagen ist unter technischen und Kostenaspekten an die Grenzen seiner Leistungsfähigkeit gestoßen. Ursächlich verantwortlich hierfür ist vor allem die mangelnde Verfügbarkeit unterstützender moderner Paradigmen, sowie die sich daraus unmittelbar ergebende Nichtverfügbarkeit von geeigneten Softwarelösungen.

Die als grundlegende Voraussetzung für einen modernen Publikationsprozess dienende Datenstruktur für Dokumente ist bereits seit Mitte der 80er Jahre entwickelt und normiert: logisch ausgezeichnete Dokumente. Sie verarbeitende Systeme sind seit dieser Zeit aber nur in geringer Anzahl entwickelt worden. Zudem tragen diese Systeme zum einen meist den Nachteil in sich, speziell für wenige spezifische Klassen von logisch ausgezeichneten Dokumenten entwickelt worden zu sein. Zum anderen ist es in diesen Systemen nicht möglich, in freier Weise Designspezifikationen zur Verarbeitungssteuerung durch dafür ausgebildete Personen entwerfen zu lassen. Es kommt hinzu, dass die besonderen Anforderungen, die sich aufgrund dem besonders in letzter Zeit aufkommenden Wunsch nach einem Multiple Product Publishing ergeben, mit diesen Systemen nicht in ausreichendem Maße erfüllt werden können.

Die sich aus oben genannten Punkten für diese Arbeit ergebende Aufgabenstellung ist deshalb zunächst die Entwicklung einer Designspezifikationsmethodik, die technisch in der Lage ist, die modernen Anforderungen an eine derartige Methodik umfassend erfüllen zu können und zugleich einfach an neu aufkommende Anforderungen anpassbar ist. Ein dabei wichtiges Augenmerk ist auf die Berücksichtigung der Personen und ihrer bisherigen Qualifikationen zu legen, die mit Hilfe der in der Designspezifikationsmethodik gegebenen Möglichkeiten Designspezifikationen erstellen sollen: Typographen und Graphikdesigner. Sie sind ausgebildete Spezialisten im Bereich der Gestaltung und Typographie, nicht jedoch im Umgang mit komplexen Systemen, die ein intimes Wissen in Bezug auf die Programmierung von Rechensystemen erfordern.

Schließlich gilt es, für die gewählte Designspezifikationsmethodik eine softwaretechnisch umsetzbare Architektur zu entwickeln und zu realisieren, die prototypisch die Eignung des gewählten Ansatzes zeigen und als Grundlage für zukünftige Entwicklungen dienen kann.

Kapitel 4

Die objektorientierte Designspezifikationsmethodik [em:]

In Kapitel 3 dieser Arbeit wurden Erstellungsszenarien für Designspezifikationen untersucht. Leitbild bei dieser Untersuchung war vor allem das klassische Vorgehen in Verlagen bei der Erstellung von Publikationen, dessen Analyse deutliche Schwachpunkte aufgedeckt hat. Verantwortlicher Kernpunkt für diese Schwachpunkte ist hauptsächlich die mangelnde Verfügbarkeit von digitalen Systemen zur Unterstützung des Publikationsprozesses. Ziel dieses Kapitels ist deshalb die Entwicklung einer Designspezifikationsmethodik, die als Basis eines digitalen Systems zur Unterstützung des digitalen Publikationsprozesses dienen kann. Ihr Wirkungsbereich soll sich aber nur auf den Zentralbereich des digitalen Publikationsprozesses, die Steuerung der Kern- und der erweiterten Aufgaben von Dokumentformatierern, erstrecken.

An dieser zentralen Stelle ist es wichtig zu betonen, dass die in dieser Arbeit entwickelte Designspezifikationsmethodik [em:] nicht zur Erstellung von Designspezifikationen mittels eines normalen Texteditors verwendet werden soll. Dies würde keinesfalls dem übergeordneten Ziel dieser Arbeit, der Erstellung eines benutzerfreundlichen Designspezifikationssystems, entsprechen. Aus diesem Grund wird in Kapitel 4 auch nur auszugsweise und vereinfacht auf die Formalismen zur textuellen Erstellung von [em:] Designspezifikationen eingegangen. In Ergänzung hierzu wird statt dessen in Kapitel 6 ausführlich die Verwendung der hier eingeführten Designspezifikationsmethodik zur Erstellung von Designspezifikationen mittels des graphischen Designeditors [de:] vorgestellt.

4.1 Überblick

4.1.1 Das System [es:]

Ziel dieser Arbeit ist die Entwicklung eines benutzerfreundlichen, mächtigen und flexiblen Systems zur Designspezifikation und Formatierung logisch ausgezeichneter Dokumente, das den

Namen [es:] trägt. Ausgangspunkt hierfür ist zunächst die Entwicklung des Herzstücks eines derartigen Systems, einer geeigneten Methodik zur Erstellung von Designspezifikationen.

Die hier entwickelte Methodik trägt den Namen [em:]. Sie unterscheidet sich von anderen verfügbaren Designspezifikationen und Stylesheet-Ansätzen in mehreren Punkten. Hervorzuheben ist an dieser Stelle im Besonderen ihre systematische Herleitung und Entwicklung anhand objektorientierter Methoden sowie ihre Ausrichtung auf die Personengruppe der *Nichtprogrammierer* als größte potentielle Anwendergruppe.

Aufbauend auf dieser Methodik wird ein interaktives System entwickelt, das aus drei Komponenten besteht (siehe Abbildung 4.1):

- [de:], einem interaktiv graphischem Designeditor zur Erstellung von Designspezifikationen gemäß der [em:]-Methodik und dem Paradigma *Design by Example*.
- [pe:][1], einem Designprozessor zur Umsetzung einer vorgegebenen Designspezifikation für ein Dokument in eine ebenfalls im Rahmen dieser Arbeit entwickelte Designstruktur.
- [ef:], einem Dokumentformatierer und Betrachter zur Formatierung und Bildschirmanzeige bzw. Druckaufbereitung von Designstrukturen.

4.1.2 Das Paradigma *Design by Example*

Ein zentraler Gesichtspunkt im Rahmen dieser Arbeit ist das Paradigma *Design by Example*. Es bezeichnet die Vorgehensweise, Designspezifikationen für logisch ausgezeichnete Dokumente nicht wie in anderen Systemen üblich direkt anhand der Regeln der Dokument Typ Definitionen für gewünschte Klassen von Dokumenten zu erstellen.

Statt dessen werden zur Erstellung von Designspezifikationen im System [es:] repräsentative, logisch ausgezeichnete Beispieldokumente verwendet. Durch dieses Vorgehen wird es zunächst möglich, den Kontext, in dem sich logische Objekte befinden, für ihre Verarbeitung zu berücksichtigen. Weiterhin ermöglicht dieses Vorgehen die Erstellung von Designspezifikationen für Dokumente, für die keine Dokument Typ Definition existiert; ein Ansatz, der gerade mit der Verbreitung von XML immer wichtiger wird. Schließlich ermöglicht dieses Paradigma, wie sich zeigen wird, eine vollkommen neue Art von Benutzungsoberfläche für Werkzeuge zur Erstellung von Designspezifikationen, in der Designregeln graphisch mit Hilfe von *Drag & Drop* Operationen wie mit einem Baukastensystem zusammengestellt werden können. Weiterhin werden einzelne Designregeln nicht mehr isoliert definiert, sondern in Zusammenhang mit anderen Regeln, die funktional zusammengehörig sind (z.B. alle Regeln, die für die Formatierung einer Liste verantwortlich sind), erarbeitet.

[1] Vom englischen processing engine.

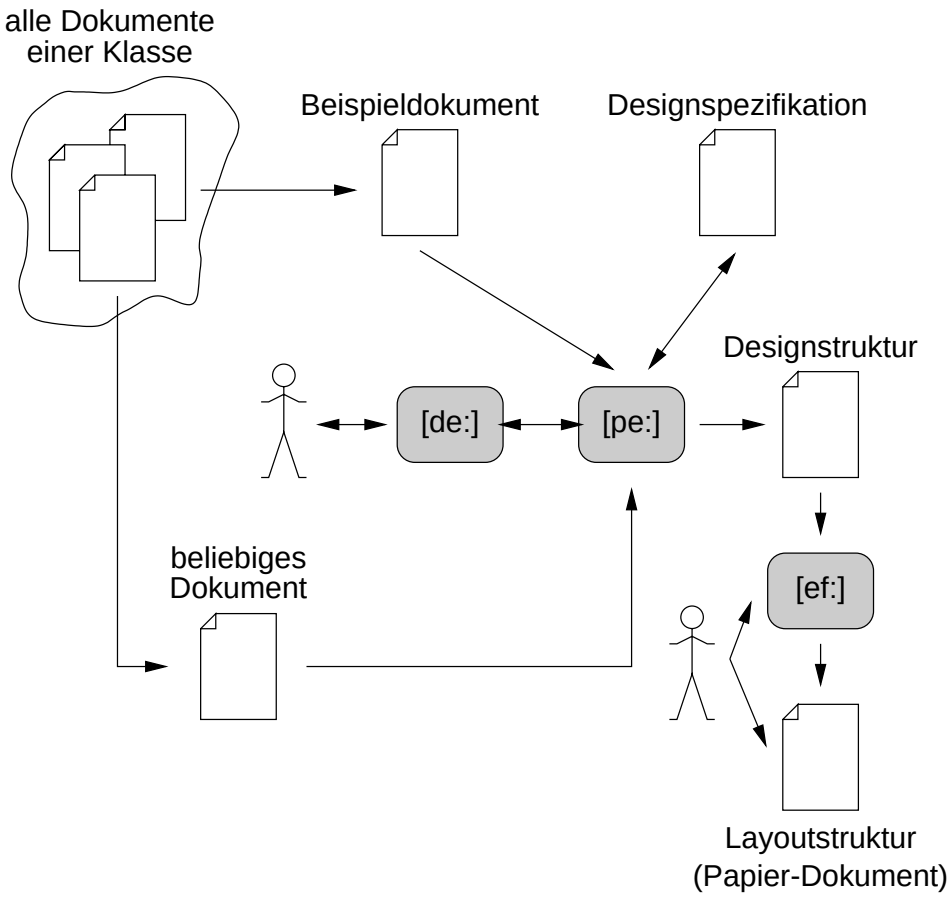

Abbildung 4.1: Grobarchitektur des [es:] Systems: Aus der Dokumentklasse, für die eine Designspezifikation erstellt werden soll, wird ein repräsentatives Dokument ausgewählt. Basierend auf dem gewählten Beispieldokument erstellt ein Graphikdesigner exemplarisch mit Hilfe des graphischen Werkzeuges [de:] die gewünschte Designspezifikation. Die dabei zur Ausführung von Berechnungen verwendete Komponente [pe:] dient später weiterhin zur Übersetzung eines beliebigen Dokumentes der vorgegebenen Dokumentklasse in die Designstruktur, die mit Hilfe des Formatierers [ef:] formatiert und visualisiert werden kann.

4.1.3 Designspezifikation und kontextabhängige Designregeln

Der Dokumentbegriff in dieser Arbeit umfasst die Bereiche logische Struktur, Daten und Layout. Die logische Struktur und die Daten sind dabei, um vom Menschen klar erkannt zu werden, von einem geeigneten Layout abhängig.

Um die logischen Paragraphen eines Dokumentes kenntlich zu machen wurde beispielsweise im vorliegenden Dokument ein Layout gewählt, das die erste Zeile beinahe eines jeden Paragraphen einzieht. Wäre dies nicht der Fall und würde Paragraph an Paragraph ohne ein visuelles, layouttechnisches Paragraphbeginn- oder -endemerkmal aneinandergereiht werden, wäre die vom Autor intendierte Paragraphgliederung vom Leser nicht mehr zu erkennen.

Den einzelnen logischen Objekten des Dokumentes muss also ein geeignetes Layout zuge-

wiesen werden. So könnte man ein logisches Objekt `Überschrift` als linksbündigen Paragraphen (für den Fall, dass dieses länger als eine Zeile ist) in der Schrift *Times bold* in der Größe 18 Punkt setzen. Um dies zu erreichen muss eine Verknüpfung zwischen dem logischen Objekt Überschrift und der gewünschten Formatierung hergestellt werden. Dies geschieht in [em:] mit Hilfe von Designregeln, die in einer Designspezifikation zusammengefaßt werden.

Bei diesem einfachen Verfahren der bedingungslosen Zuweisung von Formatierungsinformation an logische Objekte können die einzelnen logischen Objekte einer Klasse aber nicht unterschiedlich behandelt werden. Dies ist sicher nicht im Sinne der Anwender, die, gemäß anerkannten Konventionen in der Typographie, z.B. den ersten Paragraphen nach einer Überschrift ohne einen Einzug in der ersten Zeile setzen wollen.

Die Lösung dieses Problemes in [em:] besteht in der Einführung von Umgebungsprädikaten für Designregeln, mit deren Hilfe sich obiges Problem lösen läßt. Für jedes logische Objekt können mehrere unterschiedliche Designregeln spezifiziert werden, die sich in der zuzuweisenden Formatierung und im zu erfüllenden Umgebungsprädikat unterscheiden. Aus allen für ein logisches Objekt verfügbaren Designregeln wird mit Hilfe eines Auswahlverfahrens aufgrund vorgegebener Kriterien und den Umgebungsprädikaten eine geeignete Designregel ausgewählt.

4.1.4 Objektorientierter Layout-Ansatz und Layoutfluss

Die Formatierung, die mit Hilfe einer Designregel einem logischen Objekt zugeordnet wird, wird unter anderem mit Hilfe sogenannter Layoutobjekte spezifiziert. Die von Layoutobjekten geleistete Funktionalität und somit ihre funktionale Semantik, ist vermutlich den meisten Menschen intuitiv zugänglich und unmittelbar vertraut. Layoutobjekte verkörpern in der [em:]-Methodik reale Objekte einfacher Funktionalität (wie z.B. das horizontale oder vertikale Anordnen von Objekten).

Zur Erzielung komplexerer Funktionalitäten und somit aufwendigerer Layouts ist es möglich, Layoutobjekte beliebig zu kombinieren. Zur Feinsteuerung ihres Verhaltens besitzen Layoutobjekte je nach Klasse verschiedene Attribute, die hierzu von den Designern gesetzt werden können.

Ein Beispiel der aktuell im System implementierten Layoutobjektklassen ist zunächst die Layoutobjektklasse `Glyphs`, welche Eingabezeichen in anzeigbare Buchstaben formatiert (Realisierung der Glyphauswahl mit Attributen wie Font, Größe und Farbe). Als Ergebnis liefert ein Layoutobjekt der Klasse `Glyphs` Buchstaben in Form des Layoutelementes[2] `Glyph`. Aufgrund ihrer *initial produktiven* Eigenschaft[3] zählt die Layoutobjektklasse `Glyphs` zur Gruppe der sogenannten Quellobjekte.

Eine weitere Layoutobjektklasse stellt die Klasse `Paragraph` dar, welche Kind-Layoutelemente paragraphartig (gesteuert durch Attribute wie Absatzbreite, Zeilenabstand und Erstzeileneinzug) formatiert (Anwendung einer Zeilenumbruchsfunktion) und in Form eines `Paragraph` Layoutelementes an sein Vaterobjekt weiterreicht. Aufgrund dieser Funktionalität des

[2] Siehe Definition 10 auf Seite 16.

[3] Die `Glyphs` Layoutobjekte *erzeugen* ohne eine Eingabe anderer Layoutobjekte eine Ausgabe.

Akzeptierens und Aussendens von Layoutelementen gehört die Layoutobjektklasse `Paragraph` zur Menge der transienten Layoutobjekte. Ein `Paragraph` Layoutelement kann von seinem Empfänger-Layoutobjekt in mehrere Teile, sogenannte Layouteinheiten, zerlegt werden; dies ist z.B. beim Seitenumbruch, bei dem ein Layoutelement auf mehrere Seiten verteilt werden kann, nötig.

Eine letzte hier vorgestellte Layoutobjektklasse ist `Window`. Sie kommt zur Ausgabe von empfangenen Layoutelementen zur Anwendung und gehört deshalb zur Menge der Zielobjekte.

Mit Hilfe der auf ein logisch ausgezeichnetes Dokument angewendeten Designregeln wird ein Baum von Layoutobjekten erstellt. Die in diesem Baum hierarchisch angeordneten Layoutobjekte sind konzeptuell untereinander streng der hierarchisch Anordnung folgend mit einem *Pipelinesystem* (Rohrleitungssystem) verbunden. Durch diese Anordnung von durch Rohre verbundenen Layoutobjekten in einem Baum kann sich ein sogenannter Layoutfluss von Layoutelementen ergeben. Der Layoutfluss wird von Quellobjekten erzeugt, von transienten Objekten akzeptiert, verarbeitet und weitergereicht, bis schließlich ein Zielobjekt erreicht wird, das die Anzeige auf dem Bildschirm, die Ausgabe auf einem Drucker oder auch das Schreiben einer PostScript-Datei ausführt.

Dieser objektorientierte flussbasierte Ansatz wurde aus verschiedenen Gründen gewählt. Ausschlaggebend ist das Ziel, die größtmögliche Flexibilität bei der Designspezifizierung zu gewährleisten und sich soweit als möglich unkomplizierte Erweiterungsoptionen offenzuhalten. So ließe sich durch Hinzufügen neuer Layoutobjekte der Umfang der Designspezifikationsmethodik relativ einfach erweitern, sobald die Notwendigkeit dafür besteht. Ein weiterer Grund liegt in der großen Mächtigkeit der Layoutmöglichkeiten, die auf der *beliebigen* hierarchischen Kombinierbarkeit von Layoutobjekten innerhalb des Layoutobjektbaumes beruht – eine Eigenschaft, die in keinem anderen Dokumentverarbeitungssystem gegeben ist. Dies macht die Spezifizierung von typographischen Spezialitäten, wie beispielsweise einer Tabelle innerhalb einer Tabelle, zu einem Kinderspiel.

4.2 Layoutobjekte, Designstruktur und *Pipeline*-Metapher

Die Designspezifikationsmethodik [em:] stellt unter anderem eine Methode zur Beschreibung dar, wie das Layout von logisch ausgezeichneten Dokumenten (Layoutstruktur) berechnet werden soll. Es ist an dieser Stelle aber noch vollkommen offen, in welcher Form diese Beschreibung der Layoutberechnung (des Formatierprozesses) erfolgen soll. Durch die in Kapitel 3 erarbeitete Aufgabenstellung ist lediglich vorgegeben, dass die Methode funktional umfassend, einfach erweiterbar und auch für *Nichtprogrammierer* geeignet sein soll. In diesem Abschnitt über Layoutobjekte widmen wir uns deshalb zunächst ausschließlich der Entwicklung einer Beschreibungsmethodik für das Layout von konkreten Dokumenten, die auf sogenannten *Layoutobjekten* basieren wird. Die Anwendung der hier erarbeiteten Ergebnisse im Rahmen von Designspezifikationen erfolgt dann später in einem eigenen Abschnitt in dieser Arbeit.

4.2.1 Herleitung der Idee der Layoutobjekte und der Designstruktur

Die prinzipielle Annahme bei der Herleitung der Layoutobjekte ist, dass ihre späteren Anwender hochgradig im Prozess der Designspezifikation geübt sind, nicht jedoch im Umgang mit komplexen Programmiersystemen. Gegenwärtig verfügbare Systeme zur Designspezifikation sowie die darin verwirklichten Ansätze werden dieser Annahme jedoch nicht gerecht. Ihr gegenwärtiger Stand kann mit dem der früh verfügbaren Programmiersprachen (z.B. Assembler) gleichgesetzt werden, der beim Vergleich mit heute verfügbaren objektorientierten Programmiersprachen als sehr benutzerunfreundlich bezeichnet werden kann [11]. Aus diesem Grund wird im Folgenden eine neuartige objektorientierte Möglichkeit vorgestellt, mit deren Hilfe einfach und effizient Layout spezifiziert werden kann.

Zur Verbesserung der bisher nicht zufriedenstellenden Situation bei der Beschreibung des Layouts von Dokumenten wird ein Ansatz vorgeschlagen, der sich von den in Definition 10 eingeführten Layoutelementen ableitet, mit deren Hilfe in Form einer baumartigen Anordnung (der Layoutstruktur) statisch das Layout von konkreten Dokumenten festgehalten wird.

Geleitet von einem objektorientierten Analyseansatz [16] werden die in der Realität vorkommenden Layoutelemente wie z.B. Paragraph, Buchstaben, Bildschirm und Papier, die den zukünftigen Anwendern der Designspezifikationsmethodik [em:] intuitiv vertraut sind, abstrahiert und für die Designspezifikationsmethodik [em:] *indirekt* in Form von sogenannten *Layoutobjekten*[4] modelliert. Das Adverb indirekt ist für die Modellierung deshalb nötig, da, wie sich zeigen wird, Layoutobjekte in der Designspezifikationsmethodik [em:] nicht direkt die Gegenstücke von Layoutelementen sind. Vielmehr sind Layoutobjekte *produktive* Objekte, die Layoutelemente produzieren können. Durch diese Vorgehensweise der Modellierung stehen die in der Realität vorhandenen Konzepte und Mechanismen, die das Layout von Dokumenten verursachen, in der Designspezifikationsmethodik [em:] zur Verfügung.

Da man in einer Designspezifikationsmethodik aber nicht nur statisch das Layout eines Dokumentes beschreiben will, sondern hauptsächlich den Weg, wie man es berechnen kann, benötigt man auch eine Möglichkeit, dies auszudrücken. Bei der Modellierung der Layoutelemente durch die Layoutobjekte wird deshalb nicht nur der statische Aspekt der Layoutelemente, *fertiges* Layout zu beschreiben, in die Designspezifikationsmethodik [em:] übernommen. Zusätzlich und als wesentlicher Aspekt wird auch der dynamische Aspekt, wie Layoutelemente ihre Gestalt erhalten (die Layoutberechnung), bei der Modellierung der Layoutobjekte mit berücksichtigt.

Dieser zweite dynamische Aspekt der Berechnungsbeschreibung ist zwar unmittelbar mit dem statischen Aspekt der Layoutobjekte verknüpft, kann aus ihnen aber nur indirekt abgeleitet werden; schließlich gibt es funktional betrachtet mehrere Möglichkeiten, um den Eindruck ein und desselben Layoutelements hervorzurufen. Layoutobjekte tragen somit eine funktionale Semantik, sie spezifizieren klar festgelegte Formatierfunktionen. Sie können allgemeinen

[4] Es ist an dieser Stelle darauf hinzuweisen, dass die in DSSSL eingeführten *Flow Objects* aufgrund einer zu diesem Ansatz ähnlichen Auffassung analysiert worden sind. Wie sich zeigen wird, ist die bei der Modellierung der einzelnen in DSSSL vorgeschlagenen Flow Objects verwendete Vorgehenswiese jedoch zur Schaffung einer universellen Designspezifikationsmethodik nicht konsequent genug.

objektorientierten Ansätzen folgend in ihrem Verhalten über Attribute gesteuert werden. Layoutobjekte sind also Objekte, die auf einem sehr hohen benutzerangepassten Niveau deklarativ die Aspekte des Layouts von Dokumenten beschreiben können.

Definition 28 (Layoutobjekt) *Layoutobjekte dienen in der Designspezifikationsmethodik [em:] zur deklarativen Spezifikation der Anwendung von attributierten Formatierfunktionen. Sie können als Ergebnis der Anwendung der durch sie spezifizierten Formatierfunktion Layoutelemente produzieren. Ihre Verwendung in einer Designstruktur beschreibt bei einigen Layoutobjekten*[5] *die Anwendung der durch sie spezifizierten Formatierfunktion auf die Ausgabe*[6] *ihrer potentiell vorhandenen Kind-Layoutobjekte. In anderen Fällen erzeugen Layoutobjekte Layoutelemente ausschließlich aufgrund der Wertsetzung für die Attribute der durch sie verkörperten Formatierfunktion. Zur Feinsteuerung der Layoutobjekte stehen Attribute zur Verfügung, die von den Nutzern gesetzt werden können.*

Definition 29 (Designstruktur) *Die Designstruktur eines konkreten Dokumentes ist eine hierarchische baumförmige Struktur, die Layoutobjekte in beliebiger Kombination enthält. Sie spezifiziert statisch einen Formatierprozess.*

Die bedeutenden Auswirkungen der beliebigen Kombinierbarkeit von Layoutobjekten können an dieser Stelle der Arbeit noch nicht sinnvoll erklärt werden. Erwähnenswert sind an dieser Stelle aber trotzdem bereits zwei daraus resultierende Eigenschaften: Zum einen können Layoutkombinationen spezifiziert werden, die in anderen Systemen bisher nicht ausgedrückt werden können. Technisch betrachtet wird in der Designspezifikationsmethodik [em:] die in anderen Systemen existierende Aufteilung von Layoutelementen in die Kategorien *inlined* und *displayed*[7] aufgehoben. Dies ist eine Designentscheidung, die den Anwendern der Designspezifikationsmethodik [em:] extreme Freiheiten beim Entwurf von Designspezifikationen gibt. Zum anderen können aber auch Situationen im spezifizierten Layout entstehen[8] (z.B. ein Glyph in einem Glyphen), die auf den ersten Blick kurios erscheinen, durch einfache Konventionen aber elegant und benutzerfreundlich aufgelöst werden können.

Der Zusammenhang zwischen Layoutelementen und Layoutobjekten bzw. der Layoutstruktur und der Designstruktur wird in Abbildung 4.2 verdeutlicht. Layoutelemente beschreiben in einer Layoutstruktur statisch das schon berechnete Layout eines konkreten Dokumentes. Die Layoutelemente können dabei in mehrere zusammengehörige Layouteinheiten aufgebrochen worden sein. Die Layoutstruktur und die darin enthaltenen Layoutelemente sind das Ergebnis der Berechnungen eines Formatierprozesses. Die Designstruktur hingegen beschreibt mit Hilfe der in ihr enthaltenen Layoutobjekte einen Formatierprozess, aus dem sich nach seiner Ausführung eine Layoutstruktur ergibt: Jedes Layoutobjekt hat in sich das Wissen gespeichert, das es ihm ermöglicht (Nutzung der durch es spezifizierten Formatierfunktionen bei geeigneter Setzung der Attribute hierfür), seine relevanten Teile in der Layoutstruktur zu berechnen.

[5] Siehe Seite 98, Einführung der transienten Objekte und der Zielobjekte.

[6] Siehe Seite 99, Der Layoutfluss.

[7] Für Näheres zu inlined und displayed Layoutelementen im Kontext des DSSSL Standards siehe Seite 68.

[8] Da, wie erst später in dieser Arbeit vorgestellt wird, Designstrukturen aufgrund gegebener logisch ausge-

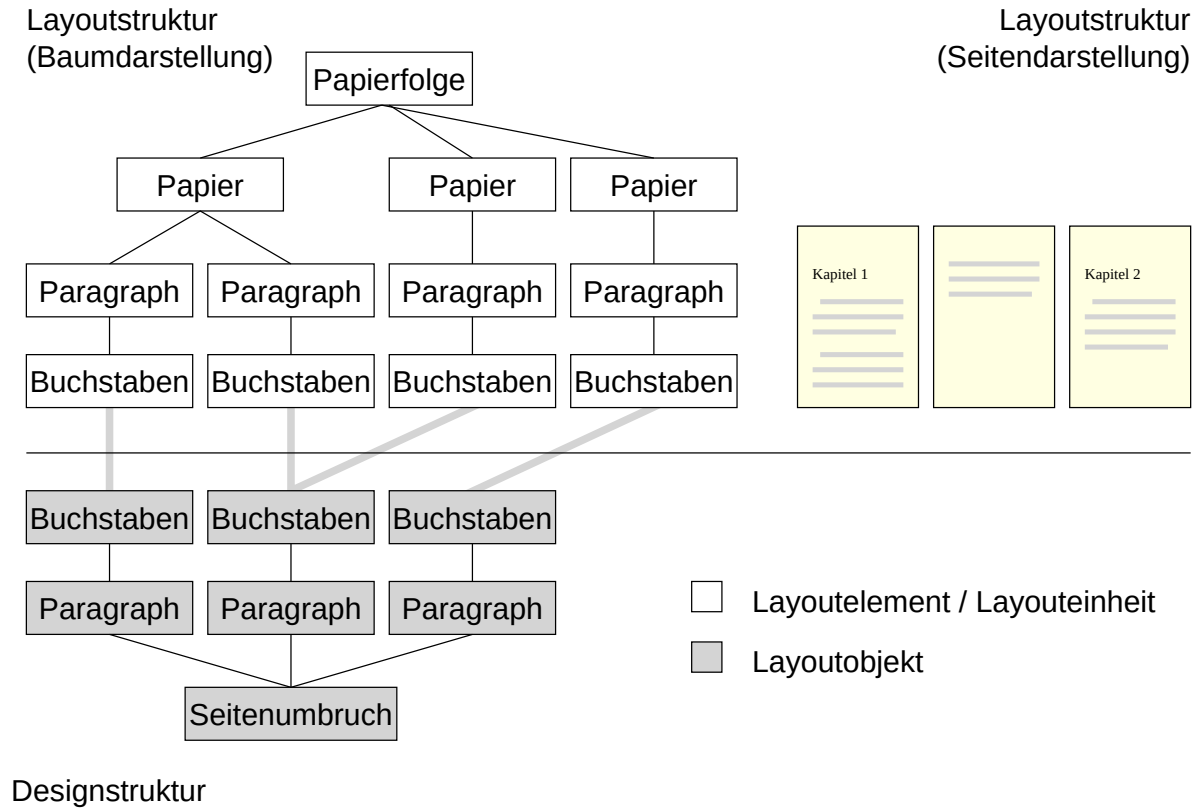

Abbildung 4.2: Gegenüberstellung Layoutstruktur und Designstruktur

4.2.2 Herleitung der Kriterien für eine Layoutobjekt-Basismenge

Bereits im Rahmen der Definition zu Layoutobjekten wurden als ad hoc Beispiele für Layoutobjekte Paragraph, Buchstabe, Bildschirm und Papier genannt. Es stellt sich aber sofort die Frage, welche weiteren Layoutobjekte im Sinne dieser Definition prinzipiell existieren und ob diese Menge als Grundlage für die universelle Beschreibung des Layouts von Dokumenten optimal geeignet ist, oder aufgrund weiterer Kriterien beeinflusst werden muss.

Zum Einstieg in eine Lösung dieser Fragestellung dient Abbildung 4.3. Zentral ist dort ein konkretes Dokument abgebildet, für das auf der linken und rechten Seite eine *naheliegende* Sequenz von mentalen Objekten (siehe Definition 30) vorgeschlagen wird, die ein Nutzer des Dokumentes (natürlich neben dem eigentlichen Inhalt des Dokumentes) beim Lesen des Dokumentes erkennt. Mentale Objekte sind dabei wie folgt festgelegt[9]:

Definition 30 (Mentale Objekte) *Mentale Objekte sind durch Dokumentnutzer beim Lesenlernen verinnerlichte, allgemein verfügbare Konzepte, die Dokumentbestandteile repräsentieren*

zeichneter Dokumente *automatisch* gemäß einer gegebenen Designspezifikation berechnet werden, können sich nur schwer vorhersagbare Kombinationen von Layoutobjekten in Designstrukturen ergeben. Diese müssen aus Gründen einer an die Methodik zu stellenden Robustheitsforderung jedoch in jedem Fall verarbeitbar sein.

[9] Siehe auch [11].

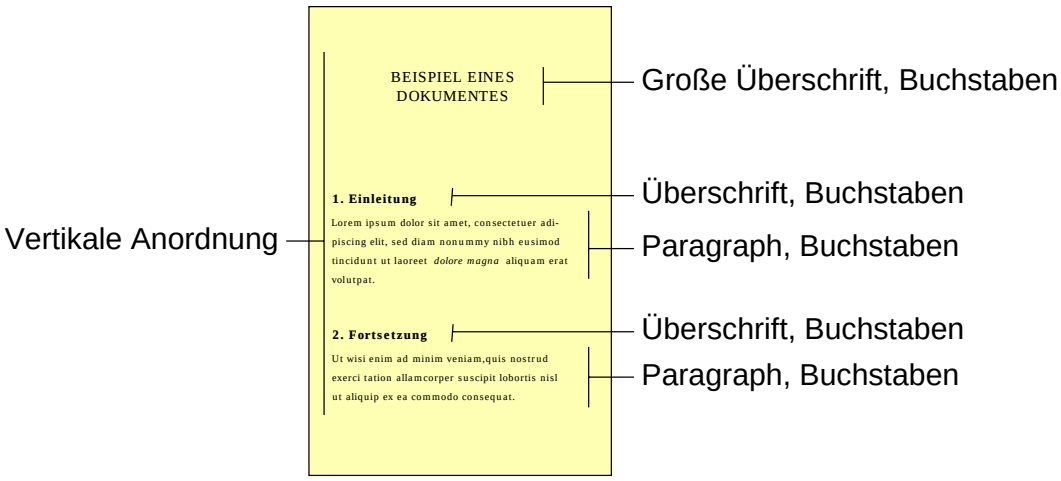

Abbildung 4.3: Mentale Objekte eines konkreten Dokumentes

(z.B. Überschriften, Paragraphen, Listen). Mentale Objekte können rekursiv verschachtelt sein (z.B. Absätze in einer Tabelle).

Es ist naheliegend, dass mit Hilfe der derart definierten mentalen Objekte eine erste sinnvolle Menge von Layoutobjekten festgelegt werden kann: Die Wahrnehmung der in einem Dokument enthaltenen strukturierten Daten wird ja direkt über die mentalen Objekte erreicht.

Vergleicht man jedoch die in Abbildung 4.3 gegebenen mentalen Objekte große Überschrift, Überschrift und Paragraph miteinander, fällt auf, dass man sie bezüglich ihrer Grundfunktionalität zusammenfassen kann. Ihre prinzipielle Aufgabe ist der Zeilenumbruch der in ihnen enthaltenen Buchstaben. Sie unterscheiden sich lediglich in der Ausrichtung der in ihnen enthaltenen Buchstaben, einer Eigenschaft, die in objektorientierten Ansätzen über die Attribute von Objekten gesteuert werden kann.

Ein erster Schritt zur Auffindung einer *verbesserten* Grundmenge von Layoutobjekten ausgehend von den mentalen Objekten kann durch eine *Generalisierung* der mentalen Objekte erfolgen. Diese Generalisierung ist ein analytischer Prozess, der zum einen für neu in die Layoutobjekt-Basismenge aufzunehmende Layoutobjekte überprüft, ob das neue Layoutobjekt nicht durch ein allgemeineres, bereits in der Basismenge enthaltenes Layoutobjekt bei geeigneter Parametersetzung realisiert werden kann. Zum anderen wird dabei aber auch überprüft, ob bereits in der Basismenge enthaltene Layoutobjekte durch das neue ersetzt werden können. Anstatt der drei Layoutobjekte große Überschrift, Überschrift und Paragraph ergibt sich nach der Generalisierung nur noch ein Layoutobjekt Paragraph, das durch eine geeignete Setzung von Parametern die Effekte der anderen mentalen Objekte mit abdecken kann.

Bei Durchsicht des DSSSL-Standards [30] und der darin definierten Objekte (Flow Objects) fällt auf, dass dort offensichtlich ebenfalls ein derartiges Verfahren der Generalisierung bei der Auswahl der definierten Objekte angewandt wurde.

Allgemein ist aber die Menge der generalisierten Objekte und somit auch die Menge der im DSSSL-Standard definierten Objekte bezüglich weiterer Kriterien *nicht* für eine universelle Designspezifikationsmethodik geeignet. So sind dort beispielsweise Objekte vorhanden, die

zwar nicht bezüglich ihrer gesamten Funktionalität identisch sind, aber zumindest in Bezug auf Teilfunktionalitäten. Aus diesem Grund entsteht somit zum einen erneut ein gewisser Grad an Redundanz in der Methodik. Zum anderen ist aufgrund der in den bereits definierten Objekten vorhandenen Redundanz zu vermuten, dass diese Teile auch in weiteren Objekten einer auf derartigen Objekten basierenden Methodik explizit benötigt werden, aber nicht als eigene Einheit zur Verfügung stehen.

Die derart festgelegte Basismenge enthält *zu komplexe* Layoutobjekte, die sehr nah an real vorkommenden typischen Designproblemen ausgerichtet sind. Zur Spezifikation von neuen Designanforderungen reichen sie aufgrund dieser hohen Spezialisierung aber nicht mehr aus, ein derartiger Ansatz erreicht zu schnell seine Grenzen. Auch ist darauf hinzuweisen, dass in DSSSL nicht beliebige Kombinationen von Flow Objects zulässig sind. Aus diesem Grund können nicht alle gewünschten Layoutkombinationen spezifiziert werden.

Zur fundierten Herleitung einer Basismenge von Layoutobjekten reicht also die einfache Generalisierung von möglichen Objekten alleine noch nicht aus. Ein systematischer Ansatz zur Festlegung der gesuchten Basismenge von Layoutobjekten, der an dieser Stelle nun vorgeschlagen wird, besteht zunächst aus zwei Kriterien, die jedes Layoutobjekt in der Basismenge an sich erfüllen muss, und schließlich aus zwei weiteren Kriterien, die die gesamte Basismenge erfüllen muss.

Kriterien einer Layoutobjekt-Basismenge

1. **Aufgabenorientiertheit:**
   Jedes Layoutobjekt der Basismenge spezifiziert eine klar definierte Formatierfunktion, die im Anwendungsbereich einer Designspezifikation *natürlich*[10] ist. Die Formatierfunktion ist in der Regel attributiert und kann über geeignete Wertsetzungen an spezielle Aufgabenstellungen individuell angepasst werden.

2. **Einfachheit:**
   Die von den Layoutobjekten der Basismenge spezifizierten Formatierfunktionen sind so elementar wie möglich, damit sie bei der Spezifizierung von komplexeren Formatierfunktionen wiederverwendet werden können.

3. **Minimalität:**
   Keine durch ein Layoutobjekt der Basismenge spezifizierte Formatierfunktion kann durch eine geeignete Kombination aus den anderen Layoutobjekten der Basismenge konstruiert werden.

4. **Vollständigkeit:**
   Mit Hilfe aller in der Layoutobjekt-Basismenge enthaltenen Layoutobjekte muss es möglich sein, neue, zum Zeitpunkt der Erstellung der Layoutobjekt-Basismenge noch unvorhergesehene Formatierfunktionen, zusammenstellen zu können.

[10] Die Natürlichkeit einer Formatierfunktion ist ein formal nicht exakt fassbarer Begriff. Die Menge der natürlichen Formatierfunktionen ergibt sich vielmehr aus dem gesammelten Wissen, das sich allgemein im Laufe der Zeit bei der Realisierung von Dokumentverarbeitungssystemen ergeben hat.

Die soeben eingeführten Kriterien an eine Basismenge von Layoutobjekten werden in Abbildung 4.4 graphisch veranschaulicht.

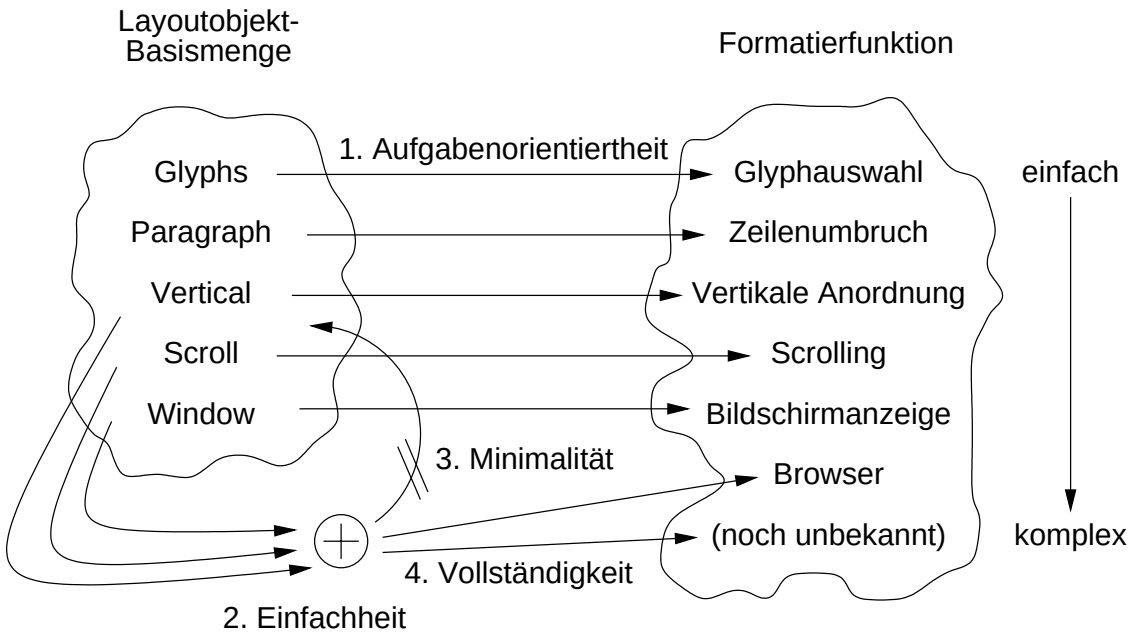

Abbildung 4.4: Kriterien für eine Layoutobjekt-Basismenge

Nach dieser Einführung der Kriterien für eine Layoutobjekt-Basismenge stellt sich unverzüglich die Frage nach deren Auswirkungen und einer Bewertung dieser Auswirkungen. Eine diesbezügliche Untersuchung muss dabei unter zwei verschiedenen Gesichtspunkten ausgeführt werden: Einerseits gilt es, die Kriterien und deren Auswirkungen bezüglich ihres Einflusses auf funktionale Aspekte bei den Möglichkeiten einer Designspezifikation zu untersuchen. Andererseits ist es aber auch nötig, die Benutzbarkeit der Layoutobjekte der Basismenge durch die späteren Anwender der Layoutobjekte zu evaluieren.

Untersuchung der funktionalen Auswirkungen der Layoutobjekt-Basismengen-Kriterien

Das erste Kriterium, die Aufgabenorientiertheit für Layoutobjekte, legt fest, dass jedes Layoutobjekt eine bei Designspezifikationen natürliche Formatierfunktion spezifizieren muss. Ein derartiges Kriterium schränkt die in der Basismenge vorhandenen Layoutobjekte von funktionalen Gesichtspunkten her betrachtet ausschließlich ein, die funktionale Ausdrucksstärke wird aber aufgrund des Ausschlusses nur *unnatürlicher* Formatierfunktionen nicht beeinflusst.

Das zweite Kriterium der Einfachheit bewirkt eine Aufsplittung von komplexen Formatierfunktionen in ihre elementaren Bestandteile, bevor diese in die Basismenge aufgenommen werden können. Die direkte Auswirkung daraus ist, dass die in der Layoutobjekt-Basismenge vorkommenden Layoutobjekte einfach wiederverwendet und sogar zu neuen, bei der Zusammenstellung der Layoutobjekt-Basismenge noch unvorhergesehenen Formatierfunktionen, zusammengestellt werden können.

Das dritte Kriterium der Minimalität bewirkt ebenfalls wie die zwei vorhergehenden Kriterien eine Verminderung der potentiell in der Layoutobjek-Basismenge enthaltenen Layoutobjekte. Durch die Substituierbarkeit der ausgeschlossenen nicht minimalen Layoutobjekte durch die Kombination anderer in der Basismenge enthaltenen Layoutobjekte ergibt sich funktional jedoch keine Einschränkung im Vergleich zur Hinzunahme der redundanten Layoutobjekte.

Das vierte Kriterium der Vollständigkeit wirkt nicht beschränkend auf die in der Layoutobjekt-Basismenge enthaltenen Objekte. Vielmehr fordert es die Eigenschaft der Basismenge, als Einheit in einer gewissen Art und Weise *universell* zu sein – eine Eigenschaft, die, wenn überhaupt, nur schwer nachgewiesen werden kann.

Untersuchung der Nutzbarkeit der Layoutobjekt-Basismenge

Die Forderung der Aufgabenorientiertheit und somit der Natürlichkeit der durch die Layoutobjekte spezifizierten Formatierfunktionen kann bei den Anwendern der Layoutobjekte eine gewisse natürliche Vertrautheit mit dem Ansatz, Layoutobjekte zur Designspezifikation zu verwenden, hervorrufen: der verwendete Ansatz ermöglicht den Benutzern in gewissen Punkten bereits von ihrer bisherigen Arbeit her bekannte und verständliche Konzepte weiter zu verwenden.

Die Einfachheit der Layoutobjekte erleichtert es den Benutzern, die Funktionalität der einzelnen Layoutobjekte zu verstehen, zu erlernen und sich ihrer bei potentiellen Anwendungsfällen während einer Designspezifikation zu erinnern.

Die Forderung der Minimalität hält die Anzahl der in der Layoutobjekt-Basismenge enthaltenen Layoutobjekte deutlich kleiner, und somit die Menge insgesamt übersichtlicher, als dies bei einem Verzicht auf diese Forderung der Fall wäre. Somit fällt es den Nutzern leichter, bei einer Designaufgabe die geeigneten Layoutobjekte zur Erreichung ihres Zieles auszuwählen und anzuwenden.

Zusammen mit der schwer zu prüfenden Forderung der Vollständigkeit ermöglichen die zuerst diskutierten drei Basismengen-Kriterien den Benutzern der Layoutobjekte, in einer kreativen Arbeitsweise Layoutaufgaben in einer spielerischen Art und Weise durch Kombination von übersichtlichen Layoutobjekten zu lösen.

4.2.3 Layoutobjekt-Basismenge für den Aufgabenbereich *Screen-Setting*

Im konventionellen Publikationsprozess, der als Produkt ein gedrucktes Buch erzeugt, bezeichnet man den Vorgang des Setzens im Englischen auch mit *Type-Setting*. Moderne Anforderungen an die Verlage erfordern aber auch die Möglichkeit der bildschirmgerechten Aufbereitung von Inhalten. Ein für diese Aufbereitung möglicher Begriff ist *Screen-Setting*. Das Screen-Setting wird, wie die enorme Verbreitung der WWW-Browser zeigt, ein technisch und kommerziell immer wichtigerer Punkt. Diese Ausgangslage dient hier als Anlass, eine Basismenge von Layoutobjekten zusammenzustellen, die die Aufgabe des Screen-Settings lösen kann.

Die Vorgabe bei der Zusammenstellung der Layoutobjekt-Basismenge ist dabei, die selben prinzipiellen Layoutmöglichkeiten erzielen zu können, wie sie mit Hilfe der logischen Auszeichnungssprache HTML[11] in den WWW-Browsern von z.B. Netscape oder Microsoft hervorrufbar sind. Vernachlässigt werden an dieser Stelle vorläufig die in HTML gegebenen Möglichkeiten der interaktiven Manipulation von Dokumentbestandteilen (z.B. Scrolling und Linking) sowie die technisch sehr anspruchsvolle Aufgabe der Behandlung von mathematischen Formeln und das bisher wissenschaftlich nur sehr vereinzelt diskutierte und modellierte Problem der Behandlung von Tabellen (siehe z.B. [52]).

Das prinzipielle Vorgehen bei der hier nur kurz beschriebenen Analyse zur Auffindung der Layoutobjekt-Basismenge ist eine Rückwärtsanalyse von HTML-Dokumenten, bei der gegebene konkrete Dokumente bezüglich der in ihnen verwendeten Formatierfunktionen analysiert werden. Die bei dieser Rückwärtsanalyse gefundenen Formatierfunktionen werden auf ihre Eignung als Grundlage für ein Layoutobjekt sukzessive jeweils mit Hilfe der Layoutobjekt-Basismengen-Kriterien auf *Zulässigkeit* (Kriterium 1 und 2), sowie die gesamte Layoutobjekt-Basismenge bei Aufnahme eines neuen Layoutobjektes auf ihre *Konsistenz* (Kriterien 3 und 4) hin geprüft. Das Ergebnis dieser Analyse ist eine überraschend kleine Menge von Layoutobjekten, deren Vollständigkeitseigenschaft in mathematischem Sinne nicht aufgezeigt werden kann. Durch den erfolgreichen Einsatz dieser Menge in praktischen Versuchen erhärtet sich jedoch offensichtlich die Behauptung, dass sie ausreichend ist.

Screen-Setting Layoutobjekt-Basismenge

Die Layoutobjekt-Basismenge enthält die in folgender Aufzählung englisch benannten acht Layoutobjekte `Glyphs`, `Picture`, `Space`, `Paragraph`, `Vertical`, `Horizontal`, `Sub-Super` und `Window`. Die von ihnen gekapselten Formatierfunktionen samt einer informellen Beschreibung ihrer wichtigsten Attribute sind in Kurzform in nachfolgender Liste aufgeführt[12]:

Glyphs Das `Glyphs` Layoutobjekt spezifiziert eine Glyphauswahlfunktion. Die zu formatierenden Buchstaben werden im Attribut `input` angegeben. Weiterhin kann unter anderem über das Attribut `font` die Schriftart, über `fontSize` die Schriftgröße und über `color` die Farbe der zu erzeugenden `Glyph` Layoutelemente festgelegt werden.

Picture Das `Picture` Layoutobjekt spezifiziert die Formatierung eines Bildes. Wichtigstes Attribut ist `name`, das den Namen der Datei spezifiziert, in der die Ausgangsdaten für das Bild abgelegt sind.

Space Das `Space` Layoutobjekt spezifiziert die Erzeugung eines horizontalen (über das Attribut `width`) und/oder vertikalen (über das Attribut `height`) Leerraums, wie er beispielsweise zwischen einer Kapitelüberschrift und dem Fließtext (vertikal) oder innerhalb einer Kapitelüberschrift zwischen der Nummerierung und dem eigentlichen Inhalt (horizontal) gewünscht ist.

[11] Aufgrund seiner technischen Entwicklung ist HTML korrekterweise weniger als logische Auszeichnungssprache, als vielmehr als Layoutsprache, zu bezeichnen.

[12] Eine vollständige Liste aller im Rahmen dieser Arbeit entwickelten Layoutobjekte ist in Anhang A zu finden.

Paragraph Das `Paragraph` Layoutobjekt spezifiziert die Anwendung einer Zeilenumbruchsfunktion. Verfügbare Attribute sind unter anderem `width` zur Steuerung der Breite der beim Zeilenumbruch erzeugten Zeilen sowie `align` zur Steuerung der Ausrichtung der Zeilen.

Vertical Das `Vertical` Layoutobjekt spezifiziert die Anwendung einer vertikalen Anordnungsfunktion auf seinen Inhalt.

Horizontal Das `Horizontal` Layoutobjekt spezifiziert die Anwendung einer horizontalen Anordnungsfunktion auf seinen Inhalt.

SubSuper Das `SubSuper` Layoutobjekt spezifiziert eine Hoch- bzw. Tiefstellungsfunktion und erlaubt damit die Realisierung von z.B. Indizes oder Fußnotenzeichen.

Window Das `Window` Layoutobjekt spezifiziert die Anzeige von formatiertem Inhalt in einem am Bildschirm eines Rechensystems dargestellten Fenster. Verfügbare Attribute sind unter anderem `height` und `width` zur Angabe der Ausmaße des Fensters.

Beispiele erzielbarer Layouteffekte

Mit Hilfe der soeben eingeführten Layoutobjekte der Layoutobjekt-Basismenge für den Aufgabenbereich Screen-Setting muss es möglich sein, alle gängigen Layoutaufgaben für die Bildschirmdarstellung von Dokumenten spezifizieren zu können. In den folgenden Beispielen werden einige Layoutprobleme sowie ihre Realisierung mit Hilfe von Ausschnitten aus Designstrukturen dargestellt.

Schriftumschaltung:
Die Umschaltung zwischen verschiedenen verfügbaren Schriftarten und analog auch die Umschaltung zwischen verschiedenen Schriftgrößen, Farben, u.s.w. kann durch die Applikation mehrerer `Glyphs` Layoutobjekte erfolgen.

Fußnotenzeichen:
Die Realisierung eines im Fließtext vorkommenden Fußnotenzeichens kann durch die kombinierte Verwendung eines `Glyphs` Layoutobjektes und eines `SubSuper` Layoutobjektes realisiert werden.

Aufzählung:
Eine unnummerierte Aufzählung kann im Wesentlichen durch die kombinierte, vernestete Verwendung des `Vertical` und des `Horizontal` Layoutobjektes erreicht werden. Ein detailliertes Beispiel dazu ist in Abbildung 4.5 dargestellt.

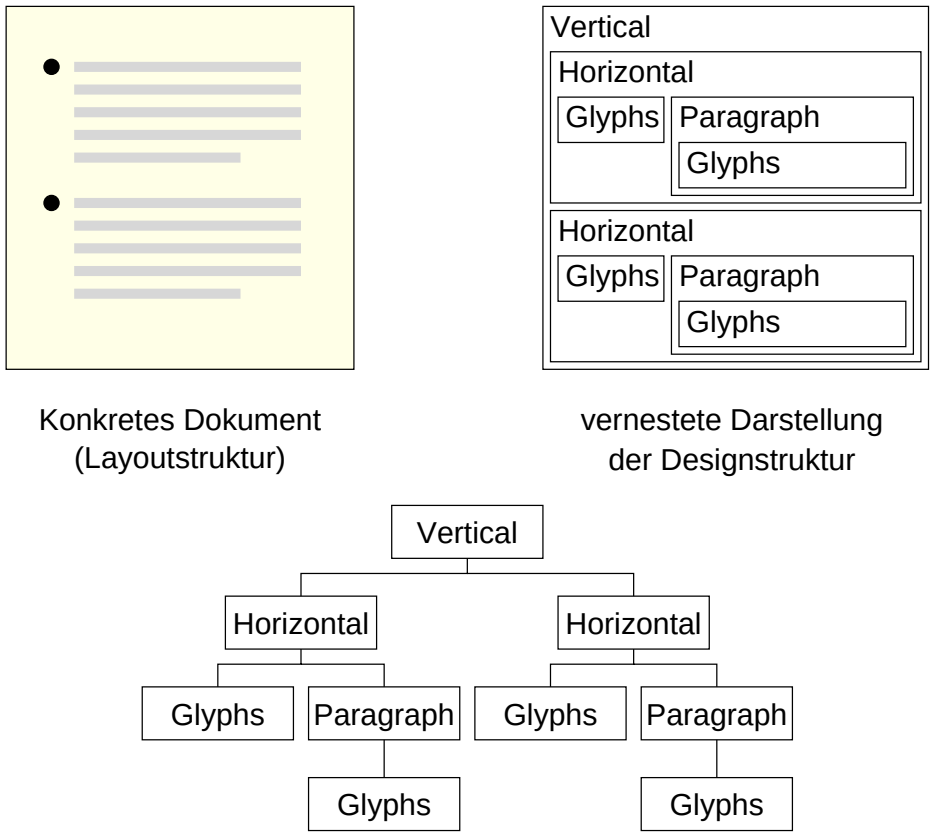

Abbildung 4.5: Designstruktur für eine Liste: Links oben in der Abbildung ist ein konkretes Dokument, das eine Liste enthält, dargestellt. Rechts daneben ist die dazugehörige Designstruktur *vernestet* dargestellt, darunter dieselbe Designstruktur in einer Baumdarstellung. Das Layout der im konkreten Dokument dargestellten Liste kann durch eine einfache Kombination von `Vertical` und `Horizontal` Layoutobjekten erreicht werden.

4.2.4 Die *Pipeline*-Metapher

Gemäß Definition 29 spezifiziert eine Designstruktur statisch einen Formatierprozess. Dies geschieht über die in ihr angeordneten Layoutobjekte. Trotz der Tatsache, dass gemäß Definition 28 die Layoutobjekte über Attribute steuerbare Formatierfunktionen repräsentieren, wurde bisher der dynamische Ablauf eines durch eine Designstruktur spezifizierbaren Formatierprozesses aber nicht beschrieben. Für die Designspezifikationsmethodik [em:] ist dies jedoch ein wichtiger Punkt, da er wesentlich zum Verständnis des Systems durch die Nutzer beitragen soll. Aus diesem Grund wird im Folgenden ein Prozessmodell eingeführt, das den Ablauf eines Formatierprozesses, der durch eine Designstruktur beschrieben wird, operativ unter Abstützung auf die Funktionalität der Layoutobjekte beschreibt.

Ziel dieses Prozessmodells ist es zunächst, den Benutzern der Designspezifikationsmethodik [em:] ein Modell für den Formatierprozess an die Hand zu geben, das aufbauend auf die

einfache Semantik der Layoutobjekte möglichst intuitiv zu verstehen und deshalb einfach zu verinnerlichen ist. Weiterhin soll dieses Modell später aber auch elegant in einer graphischen Benutzungsoberfläche umgesetzt werden können.

Quellobjekte, transiente Objekte, Zielobjekte

Wie bereits in ihrer Definition festgehalten wurde, sind Layoutobjekte *produktiv*, d.h. sie können Layoutbestandteile in Form von Layoutelementen erzeugen. Mit anderen Worten ausgedrückt kann jedes Layoutobjekt für sich genommen einen *Fluss* von Layoutelementen erzeugen.

Es liegt auf der Hand, dass Layoutobjekte, um produktiv werden zu können, auch in irgendeiner Form *Input* benötigen. Bezüglich ihres Verhaltens bei der Produktion von Layoutelementen und dem benötigten Input lassen sich die Layoutobjekte in drei Klassen einteilen. Diese Klassifizierung unterscheidet die Layoutobjekte nach dem Vorhandensein eines *Einganges* (werden Layoutelemente akzeptiert) und eines *Ausganges* (werden Layoutlemente ausgesendet).

Quellobjekte Quellobjekte haben einen Ausgang aber keinen Eingang. Sie generieren die Layoutelemente, die sie produzieren, lediglich anhand ihrer Attributierung. Typische Beispiele für Quellobjekte sind das `Glyphs` Layoutobjekt, das formatierte `Glyph` Layoutelemente produziert, oder das `Picture` Layoutobjekt, das eine Graphik aus einer Datei lädt und als `Picture` Layoutelement an seinen Ausgang weitergibt.

Transiente Objekte Transiente Objekte besitzen sowohl einen Ausgang als auch einen Eingang. Zu ihnen gehört z.B. das `Paragraph` Layoutobjekt, das seine empfangenen Layoutbestandteile (normalerweise `Glyph` Layoutlemente) in Absatzform anordnet, aber auch das `Vertical` und das `Horizontal` Layoutobjekt. Transiente Objekte senden die aufgrund ihrer Eingabe produzierten Layoutelemente am Ausgang aus.

Zielobjekte Zielobjekte sind Layoutobjekte mit einem Eingang aber ohne einen Ausgang. Sie schließen die Bearbeitung des von ihnen akzeptieren Flusses von Layoutobjekten ab, indem sie den empfangenen Fluss auf einem physikalischen Objekt sichtbar machen (z.B. das `Window` Layoutobjekt auf einem Bildschirm) oder ihn einfach verwerfen (das später noch eingeführte `Trash` Layoutobjekt).

Das Layout-Pipeline-System

Stellt man sich die Layoutobjekte einer Designstruktur als *Quellen*, *Pumpstationen* und *Verbraucher* vor und die Verbindungen zwischen ihnen als verbindende *Rohre*, so ergibt sich als

Modell für diesen Aufbau ein für die Benutzer leicht verständliches *Pipeline*-System. Im speziellen Kontext dieser Arbeit kann dieses Pipeline-System auch als Layout-Pipeline-System bezeichnet werden:

Definition 31 (Layout-Pipeline-System) *Ein Layout-Pipeline-System (auch: Pipeline-System) ist eine Designstruktur, bei der man die Quellobjekte als Quellen, die transienten Objekte als Pumpstationen (mit Pumprichtung ausschließlich von unten nach oben*[13]*), die Zielobjekte als Verbraucher und die strukturellen hierarchischen Beziehungen der Layoutobjekte als Verbindungsrohre auffasst. In einem Layout-Pipeline-System kann ein Layoutfluss*[14] *fließen.*

Der Layoutfluss

Wie in jedem Pipeline-System kann auch in einem Layout-Pipeline-System ein Inhaltsfluss fließen. Er beginnt hier normalerweise bei den Blättern (Quellen), endet an der Wurzel (Verbraucher) und durchfließt die Pumpstationen. Üblicherweise werden deswegen Quellobjekte an den Blättern einer Designstruktur lokalisiert sein, transiente Objekte im Inneren und ein Zielobjekt an der Wurzel. Im Gegensatz zu einem *echten* Pipeline-System ist es in einem Layout-Pipeline-System von wesentlichter Bedeutung, in welcher Reihenfolge die Bestandteile fließen. Folgende Definitionen regeln die Details hierzu:

Definition 32 (Aktivierung eines Layoutobjektes) *Der Begriff Aktivierung eines Layoutobjektes bezeichnet den Vorgang, ein Layoutobjekt mit der Ausführung der durch ihn verkörperten Formatierfunktion zu beauftragen. Im Rahmen seiner Aktivierung kann ein Layoutobjekt vorübergehend seine Aktivierung gezielt an seine Nachfahren weiterreichen*[15]*. Durch die Aktivierung eines Layoutobjektes werden unter Umständen Layoutelemente produziert, wozu die von untergeordneten Layoutobjekten in geordneter Folge empfangenen Layoutelemente in beliebiger Reihenfolge verwendet werden können.*

Definition 33 (Ablauf eines Layout-Pipeline-Systems) *Die geregelte Aktivierung der in einem Layout-Pipeline-System enthaltenen Layoutobjekte definiert den Ablauf des betrachteten Layout-Pipeline-Systems. Zum Start des Ablaufs wird gemäß dieser Definition immer das Wurzel-Layoutobjekt des Layout-Pipeline-Systems aktiviert, das gemäß Definition 32 die Aktivierung zur Steuerung des weiteren Ablaufs geregelt weiterreicht. Durch den Ablauf des Layout-Pipeline-Systems entsteht ein geregelter Layoutfluss*[16]*, in dem die Reihenfolge der enthaltenen Layoutelemente von signifikanter Bedeutung ist.*

[13] Unten meint im Allgemeinen Layoutbjekte, die näher an der Blättern der Designstruktur liegen, oben respektive an der Wurzel.

[14] Siehe Definition 34.

[15] Siehe hierzu die Anmerkung zu Definition 26 auf Seite 46: In komplexen Modellen muss berücksichtigt werden, dass zwischen den einzelnen Formatierfunktionen und somit auch den Layoutobjekten komplizierte wechselseitige Abhängigkeiten bestehen, die mehrere Aktivierungen eines Layoutobjektes nötig machen können.

[16] Siehe Definition 34.

Bei Definition 33 ist zu beachten, dass lediglich der Anstoß des Ablaufs eines Layout-Pipeline-Systems zentral gesteuert wird, indem das Wurzel-Layoutobjekt nach Definition zum Start des Ablaufs aktiviert wird. Der weitere Ablauf des Systems hingegen unterliegt einer *dezentralen Steuerung*, die vereint durch die Menge der im Layout-Pipeline-System angeordneten Layoutobjekte und ihrer *Aktivitätsweiterreichung* realisiert wird. Den beim Ablauf eines Layout-Pipeline-Systems entstehenden Fluss nennen wir im Folgenden Layoutfluss:

Definition 34 (Layoutfluss) *Der durch die Zusammenarbeit der Quellen, Pumpstationen, Verbraucher und Rohre entstehende Fluss in einem Layout-Pipeline-System wird Layoutfluss genannt. Er fließt von den Quellen zu den weiter oben liegenden Verbrauchern und durchfließt dabei dazwischenliegende Pumpstationen und Rohre. Der Layoutfluss transportiert Layoutelemente in einer geordneten Folge, die in den Rohren nicht verändert werden kann.*

Aufgrund der beliebigen Kombinierbarkeit der Layoutobjekte in Designsstrukturen können Designstrukturen und damit auch Layout-Pipeline-Systeme entstehen, die auf den ersten Blick wenig sinnvoll erscheinen. Dies sind Designstrukturen mit Kombinationen, in denen z.B. ein *Window* Layoutobjekt inerhalb eines *Window* Layoutobjektes (siehe Abbildung 4.6 a) oben), ein *Glyphs* Layoutobjekt ohne ein zugehöriges *Window* Layoutobjekt (siehe Abbildung 4.6 b)) oder ein *Window* Layoutobjekt ohne ein untergeordnetes Quellobjekt (siehe Abbildung 4.6 a) unten) vorkommen. Wie sich im weiteren Verlauf dieser Arbeit zeigen wird, sind derartige Kombinationen aber durchaus sinnvoll und kommen in vielen *realen* Anwendungen gezielt vor. Die Konsequenz dieser beliebigen Kombinierbarkeit ist, dass der Layoutfluss in einem Layout-Pipeline-System an jeder Stelle einen der drei im Folgenden beschriebenen Stati einnimmt:

available An dieser Stelle im Layout-Pipeline-System stehen Layoutelemente zur Verfügung. Allerdings findet sich im Folgenden in Flussrichtung kein Zielobjekt, das diese Layoutelemente akzeptieren würde. Die praktische Konsequenz für den modellierten Formatierprozess ist, dass die zur Verfügung stehenden Layoutelemente aufgrund eines fehlenden „offenen *Wasser*hahnes“ nicht ins Freie gelangen können, um sichtbar (`Window` Layoutobjekt) und deshalb von einem Leser konsumiert zu werden. (Das Pipelinesystem steht an dieser Stelle sozusagen unter Druck, ist aber verschlossen.)

possible An dieser Stelle im Layout-Pipeline-System könnten Layoutelemente transportiert werden, wenn es weiter unten Layoutobjekte gäbe, die diese produzierten. (Das Pipelinesystem ist an dieser Stelle leer.)

used An dieser Stelle im Layout-Pipeline-System sind Layoutelemente verfügbar, die später auch von einem Zielobjekt akzeptiert werden. (Das Pipelinesystem steht unter Druck; nur in diesem Fall *fließt* der Layoutfluss auch tatsächlich.)

An dieser Stelle sei darauf hingewiesen, dass auch bei einer Designstruktur-Konstellation wie in Abbildung 4.6 c), bei der ein `Glyphs` Layoutobjekt ein zugehöriges `Window` Layoutobjekt hat, der Layoutfluss nicht im Status `used` sein muss. Wie in der Realität kann in einem

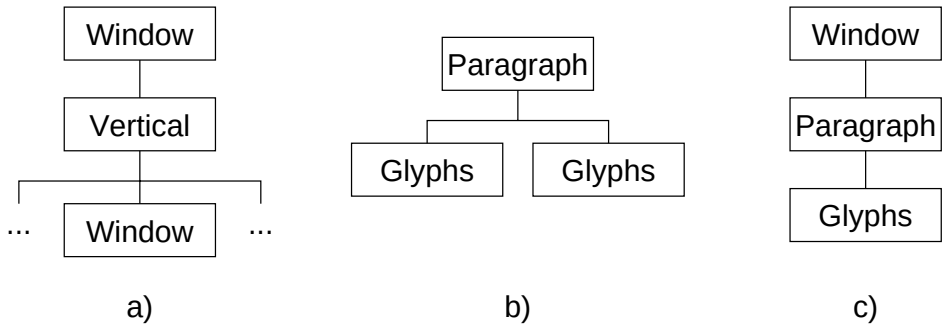

Abbildung 4.6: Ungewöhnliche Designstrukturen/Layoutflüsse: In der unter a) dargestellten Designstruktur ist ein `Window`-Layoutobjekt innerhalb eines `Window`-Layoutobjektes angeordnet; es stellt sich die Frage nach der Auswirkung des unteren `Window`-Layoutobjektes bezüglich des Layoutflusses. Unter b) ist eine Konstellation dargestellt, in der zwar Quellobjekte vorhanden sind, jedoch kein Zielobjekt; es stellt sich die Frage nach der Verfügbarkeit des erstellten Paragraphen für die Nutzer im zu dieser Designstruktur assoziierten konkreten Dokument. Die unter c) dargestellte Konstellation ist auf den ersten Blick sinnvoll und legt einen Staus des Layoutflusses von `used` nahe; aufgrund geeigneter Attributsetzung beim `Glyphs`-Layoutobjekt kann der Status jedoch auch nur `possible` sein.

Layout-Pipeline-System eine Quelle versiegt sein und folglich nicht in der Lage sein, einen Layoutfluss zu produzieren. Im Falle eines `Glyphs` Layoutobjektes ist diese Situation gegeben, wenn sein `input` Attribut der leere String ist.

Der Status des Layoutflusses an einer beliebigen Stelle in einem Layout-Pipeline-System kann also nicht in allen Fällen ausschließlich über die durch die Designstruktur gegebene Konstellation der Layoutobjekte berechnet werden, sondern ist in den Fällen, die aufgrund der statischen Konstellation `available` oder `used` als Zustand nahelegen, auch zusätzlich von der Attributierung der Layoutobjekte abhängig.

Beim Layoutfluss in einem Layout-Pipeline-System können zwei Extremsituationen eintreten, die wohl zunächst Verwirrung stiften. Dies sind das unmittelbare Zusammentreffen von zwei Quellobjekten bzw. zwei Zielobjekten (ein mittelbares Zusammentreffen, bei dem zwischengeschaltete transiente Objekte vorkommen, ändert die Situation vom Resultat her nicht). Im Falle der zwei aufeinandertreffenden Quellobjekte ist der beim oberen der beiden Layoutobjekte eintreffende Layoutfluss in der Regel `available` (bei einem versiegten Quellobjekte wäre er `possible`). Es stellt sich in diesem Fall bei einer ersten oberflächlichen Betrachtung die Frage, wie das obere Quellobjekt darauf reagieren soll. Da nach Festlegung ein Quellobjekt keinen Eingang hat, ist die Fragestellung aber einfach zu beantworten: Das obere Quellobjekt blockt den von unten ankommenden Layoutfluss einfach ab; der untere Layoutfluss ist ab diesem neuen Quellobjekt nach oben hin nicht mehr verfügbar. Im Falle der zwei aneinanderstoßenden Zielobjekte ist die Frage nach dem Verhalten des Layoutflusses ebenso einfach zu beantworten: Ein Zielobjekt hat keinen Ausgang und produziert keinen Layoutfluss nach aussen. Folglich hat das obere Zielobjekt keinerlei Information (falls Information verfügbar sein kann, dann ja nur in Form eines Layoutflusses, der in diesem Fall aber nicht existiert) über das ihm untergeordnete Zielobjekt sowie über dessen empfangenen Layoutfluss.

Aufgrund der oben getroffenen Feststellungen kann ein Layout-Pipeline-System offensichtlich immer dann in mehrere voneinander vollkommen unabhängige *einfache* Layout-Pipeline-Systeme aufgebrochen werden, wenn es hierarchisch verschachtelte Quellobjekt- bzw. Zielobjektpaare gibt. Im Falle von Zielobjekten bildet jedes Zielobjekt als Wurzel ein neues einfaches Layout-Pipeline-System, im Falle von Quellobjekten bildet das erste Layoutobjekt unterhalb eines jeden Quellobjektes die Wurzel eines neuen einfachen Layout-Pipeline-Systems, wobei in beiden Fällen weitere vernestete einfache Layout-Pipeline-Systeme abgetrennt werden müssen. Im zweiten Fall entstehen allerdings dann Probleme, wenn das betroffene Quellobjekt mehr als nur ein Kind hat: Statt einem einfachen Layout-Pipeline-System entstehen dann soviele, wie das betroffene Quellobjekt Kinder hat.

Definition 35 (Einfaches Layout-Pipeline-System) *Ein Layout-Pipeline-System wird dann als einfach bezeichnet, wenn es keine verschachtelten Quellobjekt- bzw. Zielobjektpaare beinhaltet.*

Vom Layout-Pipeline-System zur Layoutstruktur

Der Ablauf eines Layout-Pipeline-Systems beschreibt modellhaft den Ablauf eines Formatierprozesses, dessen Formatierfunktionenhierarchie statisch über eine Designstruktur von den Benutzern festgelegt wird. Das zentrale Element bei der Ablaufbeschreibung ist der Layoutfluss der Layoutelemente durch das Layout-Pipeline-System, wobei Layoutelemente in den Pumpstationen zu neuen Layoutelementen gruppiert und dabei gleichzeitig angeordnet (formatiert) werden. Im Laufe des Flusses werden Layoutelemente also in anderen Layoutelementen aufgesammelt und *gekapselt*. Am Ende eines Layoutflusses ergibt sich deshalb bei einfachen Layout-Pipeline-Systemen ein einzelnes Layoutelement, das in der Regel ein Zielobjekt sein wird.

Stellt man nun die gegebene gekapselte Sammlung von Layoutelementen in einer Baumdarstellung dar („Ausstülpung“ der vernesteten Layoutelemente) und betrachtet dabei, dass Layoutelemente evtl. von ihren Eltern in Layouteinheiten aufgeteilt worden sein können, so erhält man als Ergebnis des Ablaufes eines Layout-Pipeline-Systems eine Layoutstruktur, die nach Definition das Layout eines Dokumentes beschreibt.

Zusammenfassend ergibt sich somit, dass indirekt über die Erstellung einer Designstruktur eine Layoutstruktur spezifiziert wird. Diese Tatsache wird später in der Designspezifikationsmethodik [em:] ausgenutzt, indem mit Hilfe dieser Indirektion das Layout von Dokumenten beschrieben wird.

Abschließend ist eine interessante Feststellung bezüglich der Äquivalenz zwischen Designstrukturen und der von ihnen berechneten Layoutstrukturen hervorzuheben: In Designstrukturen, die keine Layoutobjekte beinhalten, die Layoutelemente in Layouteinheiten aufbrechen, stimmt die Struktur der Designstruktur mit der Struktur der von ihr erzeugten Layoutstruktur in vielen Fällen überein. Ein Beispiel hierzu ist in Abbildung 4.5 zu sehen, in der die Designstruktur für eine einfache Liste dargestellt ist (die nicht dargestellte Layoutstruktur hat dieselbe Struktur wie die Designstruktur).

4.2.5 Zusammenfassung

Mit dem Ziel einer Verbesserung der bisher nicht zufriedenstellenden Situation bei den Beschreibungsmöglichkeiten des Layouts von Dokumenten wurde zu Beginn von Abschnitt 4.2 eine Analyse von Verbesserungsmöglichkeiten durchgeführt. Erfolgreich wurde dabei insbesondere die Möglichkeit untersucht, den potentiellen Benutzern der Designspezifikationsmethodik [em:] bereits aus ihrer bisherigen Arbeitsumgebung intuitiv vertraute Konzepte und Objekte zur Verfügung zu stellen. Dies gelang durch eine konsequent objektorientierte Vorgehensweise, die als Ergebnis die sogenannten Layoutobjekte hervorgebracht hat. Diese Layoutobjekte spiegeln jedoch nicht in jedem Fall direkt die den Nutzern vertrauten Konzepte und Objekte wieder, sondern vielmehr systematische Zerlegungen davon. Diese Zerlegungen sind ein kritischer Punkt in der Designspezifikationsmethodik [em:], da sie zum einen wesentlich für die Verständlichkeit des Systems für die Nutzer sind und zum anderen die Funktionalität des Systems bestimmen.

Layoutobjekte sind ein Spezifikationskonstrukt, das deklarativ die Anwendung von attributierten Formatierfunktionen kapselt und modelliert. Sie können von den Anwendern von [em:] aufgrund ihrer Realitätsnähe einfach verstanden und ohne Programmierkenntnisse benutzt werden. Layoutobjekte können ohne Einschränkung beliebig miteinander kombiniert werden und so zur Erstellung beliebiger Layouts für konkrete Dokumente verwendet werden. Die in anderen Systemen vorgenommene Aufteilung von Layoutbestandteilen in inlined und displayed ist in der Designspezifikationsmethodik [em:] aus diesem Grund nicht getroffen worden.

Layoutobjekte werden in [em:] den Benutzern in Form von Basismengen angeboten, die eine für bestimmte Aufgabenbereiche ausreichende Funktionalität zur Verfügung stellen. Gewährleistet wird dies durch die vier Kriterien, die an die in einer Basismenge enthaltenen Layoutobjekte gestellt werden. Die geforderte *Aufgabenorientiertheit* sorgt für die Zurverfügungstellung von durch die zukünftigen Nutzer von [em:] tatsächlich benötigten und bisher bei ihrer Arbeit auch schon verwendeten Spezifikationsmöglichkeiten, die *Einfachheit* bewirkt eine einfache Erlernbarkeit der Layoutobjekte und die *Minimalität* hält die Basismenge klein. Die Forderung der mit Vorsicht zu bewertenden *Vollständigkeit* sichert schließlich in Zusammenhang mit der beliebigen Kombinierbarkeit von Layoutobjekten die ausreichende Funktionalität.

Die Erarbeitung der Basismenge geschieht durch einen dafür geeigneten Personenkreis und wird nicht in die Verantwortlichkeit der Endnutzer von [em:] gelegt. Sie ist ein einmaliger Arbeitsvorgang, der ein sehr genaues Wissen um die technischen Anforderungen sowohl aus dem Bereich der Satztechnik und Typographie, als auch, wie sich zeigen wird, aus dem Bereich des Dokumentformatiererbaus erfordert. Dieser Personenkreis übernimmt einmalig durch seine Analysetätigkeit bei der Erstellung der Layoutobjekt-Basismenge den Aufwand, eine benutzerfreundliche Methodik zu erstellen, die dann vielfach unter Ausnutzung der durch sie gewährleisteten Vorteile verwendet werden kann.

Das gesamte Layout eines Dokumentes kann statisch über die Kombination einer Menge von Layoutobjekten in einer Designstruktur spezifiziert werden. Die zum einfachen Verständnis der Bedeutung und Funktionsweise einer Designstruktur entwickelte Pipeline-Metapher soll, wie bereits bei den Layoutobjekten erfolgt, den Benutzern von [em:] ermöglichen, mit Hilfe von ihnen aus dem alltäglichen Leben vertrauten Konzepten technisch nicht triviale Aufgaben

spielerisch zu verstehen: In diesem Fall den dynamischen Prozess, der aufgrund einer gegebenen statischen Designstruktur das Layout eines Dokumentes berechnet.

Zur Versinnbildlichung des dynamischen Layoutprozesses werden im Rahmen der verwendeten Pipeline-Analogie Quellen, Pumpstationen, Senken und Rohre eingeführt, die zusammen einen Layoutfluss verwalten. Dieser Layoutfluss modelliert auf eine anschauliche und leicht nachvollziehbare Art und Weise den Berechnungsvorgang des Layouts eines Dokumentes. Die einfachen physikalischen Prinzipien der Bewegung des Inhalts in einem Rohrsystem von einer Quelle durch Rohre und Pumpen hin zu einer Senke wird jedermann intuitiv verständlich sein.

Selbstverständlich soll es im Rahmen der Designspezifikationsmethodik [em:] nicht die Aufgabe der Nutzer sein, Designstrukturen zur Modellierung des Layouts von konkreten Dokumenten aufgrund von vorgegebenen abstrakten Dokumenten zu entwerfen. Dieser Vorgang wird vielmehr automatisch aufgrund einer Designspezifikation erfolgen. Designspezifikationen dienen dabei zur Steuerung eines konstruktiven Ansatzes, der aufgrund von in einer Designspezifikation gegebenen Designregeln ausgehend von einem abstrakten Dokument automatisch eine Designstruktur konstruiert, die das Layout des abstrakten Dokumentes darstellt.

4.3 Logisch ausgezeichnete Dokumente

Wie schon angedeutet wurde, wird in der Designspezifikationsmethodik [em:] eine Designstruktur aufgrund von zwei steuernden Faktoren aufgebaut. Der hierbei in diesem Abschnitt zunächst vernachlässigte erste Faktor sind Designspezifikationen. Der zweite Faktor sind die logisch ausgezeichneten Dokumente, für die konkrete Dokumente erstellt werden sollen. Aus diesem Grund wird im Folgenden das Konzept der logisch ausgezeichneten Dokumente für die Designspezifikationsmethodik [em:] ausführlich untersucht und modelliert.

4.3.1 Logische Objekte und logischer Baum

Das in Abschnitt 2.1.2 Definition 5 eingeführte Konzept der logischen Objekte wird in die Designspezifikationsmethodik [em:] sinngemäß übernommen. Die in [em:] mit dem Begriff *logisches Objekt* bezeichneten Konstrukte entstehen somit aufgrund der Einführung einer hierarchischen Struktur in ein gegebenes abstraktes Dokument. Eine unmittelbare Konsequenz daraus ist, dass logische Objekte aus anderen logischen Objekten vernestet zusammengesetzt sind.

Da logische Objekte in [em:] die logischen Objekte eines gegebenen realen logisch ausgezeichneten Dokumentes modellieren, tragen auch die logischen Objekte in [em:] einen Namen. Dieser wird gleich den Namen der logischen Objekte gesetzt, die modelliert werden.

Weiterhin gibt es in [em:] ein ausgezeichnetes logisches Objekt mit dem fest vorgegebenen Namen `text`. Dieses entspricht den Textblöcken in einem logisch ausgezeichneten Dokument (z.B. #PCDATA in SGML).

Die hierarchische Struktur der logischen Objekte und somit die Vernestung der logischen Objekte bewirkt, dass die logischen Objekte die Knoten eines Baumes bilden, der die logische

Struktur eines modellierten logisch ausgezeichneten Dokumentes repräsentiert. Diesen Baum nennen wir *Baum logischer Objekte* oder auch einfach *logischer Baum*:

Definition 36 (Baum logischer Objekte / Logischer Baum) *Die durch die logische Struktur eines modellierten logisch ausgezeichneten Dokumentes verursachte hierarchische Anordnung von logischen Objekten in einem Baum heißt Baum logischer Objekte oder kürzer logischer Baum.*

4.3.2 Logische Attribute und logischer Fluss

Wie schon in 2.3.1 vorgestellt wurde, können die logischen Objekte eines logisch ausgezeichneten Dokumentes Attribute besitzen. Diese Attribute werden in der Designspezifikationsmethodik [em:] mit dem Begriff *logische Attribute* bezeichnet. Logische Attribute sind in [em:] über einen Namen ansprechbar und besitzen als Werte Daten der Sorte `String` (Zeichenketten[17]). Sie geben den Nutzern der Designspezifikationsmethodik [em:] Zugriff auf die Daten eines logisch ausgezeichneten Dokumentes.

Definition 37 (Logische Attribute) *Die logischen Objekte eines Dokumentes verfügen über logische Attribute. Diese logischen Attribute gestatten den Zugriff auf die Daten des Dokumentes, dessen Struktur die logischen Objekte modellieren. Sie tragen einen Namen, der in der Designspezifikationsmethodik [em:] mit dem Zeichen* `@` *beginnt*[18]*. Die Werte von logischen Attributen sind von der Sorte* `String`*.*
Jedes logische `text` *Objekt führt automatisch ein logisches Attribut mit Namen* `@input` *ein. Es beinhaltet den Text, für den das logische* `text` *Objekt steht. Alle anderen logischen Objekte führen die logischen Attribute ein, die sie im modellierten logisch ausgezeichneten Dokument besitzen.*

Die Kanten, die die logischen Objekte im logischen Baum miteinander in der logischen Struktur verbinden, führen den *logischen Fluss*. Dieser fließt von der Wurzel zu den Blättern des logischen Dokumentes und verdeutlicht, an welcher Stelle im logischen Baum welche durch logische Attribute eingebrachte Information (die Daten des Dokumentes) verfügbar ist.

Definition 38 (Logischer Fluss) *Der logische Fluss eines Dokumentes stellt symbolisch die Verfügbarkeit der durch die logischen Attribute der logischen Objekte des Dokumentes eingebrachten Daten dar. Wie ein* echter *Fluss fließt auch der logische Fluss von einer Quelle (Wurzel des logischen Dokumentes) zu den Senken (Blätter des logischen Dokumentes)*[19]*. Dabei transportiert er die Werte der logischen Attributen eines jeden logischen Objektes* `o` *derart, dass sie in dem ganzen Unterbaum von* `o` *verfügbar sind.*

[17] Der Standard Zeichensatz im System [es:] basiert auf ISO 8859-1.

[18] Dies schafft einen Namensraum für die logischen Attribute, mit dessen Hilfe den Nutzern der Umgang mit der [em:]-Methodik erleichtert werden soll.

[19] Im Gegensatz zum Layoutfluss in einem Layout-Pipeline-System, der im Baum von unten nach oben fließt, fließt der logische Fluss im logischen Baum also von oben nach unten.

Die Einführung von logischen Attributen und das Konzept der Verfügbarkeit ihrer Werte in einem logischen Dokumentteilbaum basiert auf den in Programmiersprachen üblichen Elementdeklarationen. Formal wird durch die Eigenschaft eines logischen Objektes `o`, ein logisches Attribut `a` mit dem (`String`-)Wert `E` zu besitzen, die Deklaration

```
String a := E
```

vorgenommen. Durch die Attributdeklaration wird der Identifikator `a` an den Wert des Ausdrucks `E` gebunden. `a` steht somit nunmehr für diesen Wert des Ausdrucks `E` als Bezeichner zur Verfügung.

Der *Bindungsbereich* der Attributdeklaration, in der `a` für den Wert `E` steht, wird auf einen gewissen Bereich begrenzt. Dies ist der gesamte logische Dokumentteilbaum, dessen Wurzel das logische Objekt `o` ist. Dieser Teilbaum gibt den sogenannten *Gültigkeitsbereich* vor, in dem auf das logische Attribut `a`, und somit indirekt auf den Ausdruck `E`, zugegriffen werden darf.

In Programmen können Identifikatoren mehrfach vereinbart und dabei mit verschiedenen Werten belegt werden. Finden diese Deklarationen vernestet statt, sind alle Bindungen bis auf die innerste *überlagert*, man spricht von *Verschattung*. Analog zu dieser Anschauung bei Programmiersprachen können auch Attributdeklarationen in der Designspezifikationsmethodik [em:] gemäß der Vernestung der logischen Objekte vorkommen. Neuere Deklarationen verschatten dann wie in gängigen Programmiersprachen die älteren Deklarationen. Eine innere Deklaration *unterbricht* jeweils den Bindungsbereich der vorherigen Deklaration.

Zur Verdeutlichung dieses Konzepts werden oft die Begriffe *Lebensdauer einer Bindung* und *Gültigkeitsbereich einer Bindung* verwendet. Die Lebensdauer einer Bindung umfaßt den gesamten Teilbaum, dessen Wurzel `o` ist. Der Gültigkeitsbereich ist der gesamte Teilbaum abzüglich der Teilbäume für *innere* Deklarationen des Attributes `a`.

4.4 Vom logisch ausgezeichneten Dokument zur Designstruktur

4.4.1 Logische Objekte und Formatierungen

In Kapitel 2 wurde festgehalten, dass Layoutelemente zur Repräsentation und Präsentation von logischen Objekten verwendet werden. In Abschnitt 4.2 wurden Layoutobjekte und ihre hierarchische Anordnung, die Designstruktur, als Möglichkeit zur Beschreibung des Layouts und somit einer Hierarchie von Layoutelementen eingeführt. Im Folgenden wird nun eine auf dieser durch die Layoutobjekte gegebene Beschreibungsmöglichkeit beruhende Methode vorgestellt, nach der für einzelne in [em:] modellierte logische Objekte zum einen ihr persönliches, *lokales* Layout[20] mit Hilfe von Spezifikationsobjekten (die auch Layoutobjekte umfassen) festgelegt wird. Dies geschieht mit Hilfe einer im Vergleich zu einer Designstruktur eines vollständigen konkreten Dokumentes *kleinen* Hierarchie von Spezifikationsobjekten, die *Formatierung*

[20] Einzelne logische Objekte können mehrfach ein lokales Layout erzeugen: Z.B. wird ein logisches Objekt `Kapitelüberschrift` Layoutelemente zum Inhaltsverzeichnis und zum Dokumentrumpf beitragen.

genannt wird. Zum anderen werden mit Hilfe einer Formatierung aber auch *globale* Auswirkungen von logischen Objekten (z.B. Fortschaltung von Zählern zur Kapitelnummerierung) festgelegt. Eine Kombination aller Formatierungen der logischen Objekte eines Dokumentes wird dann, wie später gezeigt wird, die Designstruktur zu diesem gesamten Dokument ergeben.

Bei der Untersuchung der Dokumentformatierer für graphisch ausgezeichnete Dokumente wurde aufgezeigt, dass die zur graphischen Auszeichnung eines Dokumentes verwendeten Tags eine Hierarchie von Formatierfunktionen beschreiben: Eine notwendige Voraussetzung, um ein Dokument formatiert darstellen zu können. Ein analoges Vorgehen wird deshalb in der Designspezifikationsmethodik [em:] nachgebildet. Die logischen Auszeichnungstags eines abstrakten Dokumentes implizieren einzelne Layoutobjekte bzw. hierarchische Kombinationen von Layoutobjekten, um für ein logisches Objekt ein Äquivalent zu dieser Formatierfunktionenhierarchie aufzubauen.

Die Untersuchung der Dokumentformatierer für graphisch ausgezeichnete Dokumente hat aber ebenfalls aufgezeigt, dass die graphische Auszeichnung eines Dokumentes alleine nicht ausreicht, um z.B. Nummerierungen oder Querverweise automatisch berechnen zu können. Aus diesem Grund wird es auch in der Designspezifikationsmethodik [em:] nicht ausreichen, aufgrund logischer Objekte lediglich Hierarchien von Formatierfunktionen aufzubauen. Vielmehr müssen zusätzlich auch die erweiterten Aufgaben der Dokumentformatierer beim Aufbau einer Formatierfunktionenhierarchie aufgrund eines logischen Dokumentes mit abgedeckt werden.

Zur Verdeutlichung des Aufbaus der Formatierfunktionenhierarchie und der Steuerung der erweiterten Aufgaben von Dokumentformatierern wird im Folgenden Bezug auf Abbildung 4.7 genommen, in der die logische Struktur einer Liste mit zwei Einträgen dargestellt ist (siehe hierzu auch Abbildung 4.5, in der eine mögliche Designstruktur für diese Liste dargestellt ist.).

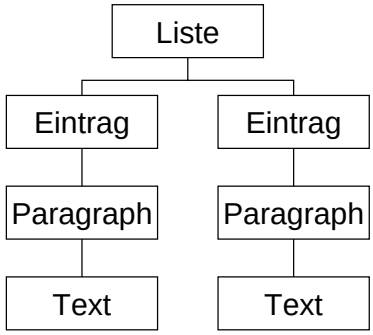

Abbildung 4.7: Logische Struktur einer Liste mit zwei Einträgen

Für viele (jedoch nicht alle) der in einem Dokument vorkommenden logischen Objekte wird eine Formatierfunktion bzw. eine Menge von Formatierfunktionen zu deren graphischer Umsetzung verwendet. Im Falle einer Liste werden z.B. in einem Dokument mit *üblichem* Layout in der Regel alle einzelnen Einträge vertikal untereinander angeordnet sein. Weiterhin wird üblicherweise für jeden Eintrag der Liste eine horizontale Kombination aus einem Listeneintragszeichen und dem eigentlichen Eintrag, der z.B. einfach aus einem Paragraphen (der Glyphen beinhaltet) bestehen kann, realisiert sein. Alle vier getroffenen Entscheidungen (vertikale Anordnung für die gesamte Liste, horizontale Anordnung für die Einträge und paragraphartige

Anordnung für den in Form von Glyphen realisierten logischen Text-Inhalt) sind dabei strikt unabhängig voneinander.

Einige der logischen Objekte eines abstrakten Dokumentes sind neben der Tatsache, dass sie Layout produzieren lassen, auch dafür verantwortlich, die erweiterten Aufgaben von Dokumentformatierern zu steuern. Im Falle einer Liste (die im aktuellen Beispiel jedoch unnummeriert ist) könnte jedes logische Objekt `Eintrag` dazu verwendet werden, einen Zähler fortzuschalten, der aufgrund des logischen Objektes `Liste` geeignet initialisiert wird.

In [em:] werden die im obigen Beispiel aufgeführten Operationen durch sogenannte *Formatierungen* nachgebildet, die lokal und unabhängig[21] voneinander für einzelne logische Objekte beschreiben, welche Auswirkungen ihre Vorkommen in einem abstrakten Dokument haben[22] und wie sie gelayoutet werden sollen. Die soeben erwähnten Formatierungen beinhalten dazu Spezifikationsobjekte:

Definition 39 (Spezifikationsobjekte) *Im Sinne der objektorientierten Programmiersprachen ist die Klasse der Spezifikationsobjekte eine Oberklasse für alle in [em:] verwendeten Objektklassen. Eine Ausnahme hierzu stellen lediglich die logischen Objekte dar, die nicht zu den Spezifikationsobjekten zu rechnen sind. Spezifikationsobjekte dienen zur Anwendungsbeschreibung der Kern- und der erweiterten Aufgaben von Dokumentformatierern. Jede individuelle Spezifikationsobjektklasse verkörpert die Beschreibung einer speziellen Funktionalität, die über die geeignete Setzung von Attributen gesteuert werden kann.*

Definition 40 (Spezifikationsattribute) *Die zur Feinsteuerung der Spezifikationsobjekte verwendeten Attribute heißen Spezifikationsattribute. Sie können von einer der Sorten* `Bool`, `Number`, `Length` *oder* `String` *sein. Ein Spezifikationsattribut muss [em:]-weit und somit spezifikationsobjektübergreifend eindeutig mit einer konstanten Sorte verwendet werden.*

Insbesondere sind also nach Definition 39 alle Layoutobjekte Instanzen von Layoutobjektklassen, die von der Basisklasse der Spezifikationsobjekte abgeleitet sind. Weitere abgeleitete Objektklassen, die in [em:] für verschiedene Aufgabenbereiche Verwendung finden, werden im Folgenden in dieser Arbeit anhand von Beispielen schrittweise eingeführt.

Definition 41 (Formatierung) *Eine Formatierung ist eine beliebige hierarchische Kombination von Spezifikationsobjekten. Sie beschreibt in lokaler Form die Auswirkungen, die ein logisches Objekt zum Layout eines Dokumentes beiträgt.*

Zunächst werden im weiteren Verlauf dieser Arbeit nur die Aspekte der Formatierungen besprochen, die ausschließlich Layoutobjekte als Spezifikationsobjekte beinhalten. Erst an späterer Stelle wird dann aufgabenorientiert der Themenbereich der erweiterten Aufgaben von Dokumentformatierern in Zusammenhang mit Formatierungen und weiteren Spezifikationsobjekten besprochen.

[21] Die zunächst unabhängigen Formatierungen erreichen aber erst bei ihrer gemeinsamen Verwendung ihr Ziel, das Layout eines Dokumentes zu beschrieben.

[22] Z.B. Erhöhung der Kapitelnummer um eins aufgrund des Vorkommens eines logischen Objektes `Kapitel` in einem Dokument.

4.4.2 Layoutobjekte in Formatierungen

Im Sinne von Definition 41 werden zur Modellierung des Listenbeispiels aus Abbildung 4.7 in folgender Aufzählung vier verschiedene Formatierungen verwendet, die jeweils individuell das Layout beschreiben, das den Nutzer eines konkreten Dokumentes die logische Aufgabe der Dokumentbestandteile sowie deren Inhalt *am besten* vermittelt. Dies wird sicherlich dann erreicht, wenn die angeführten Formatierungen *üblicherweise* verwendete Layouts beschreiben, die allgemeinen Konventionen entsprechen.

- Die erste Formatierung besteht einfach aus einem `Glyphs` Layoutobjekt und legt fest, dass auf die im logischen Dokument gegebenen Buchstaben eine Glyphauswahlfunktion angewendet wird.
- Auch die zweite Formatierung besteht nur aus einem Layoutobjekt. In diesem Fall wird für die Spezifikation des Layouts eines logischen Paragraphens ein `Paragraph` Layoutobjekt verwendet.
- Die dritte Formatierung, die das Layout eines Eintrages beschreibt, ist etwas komplizierter und besteht aus zwei Layoutobjekten, einem `Horizontal` und einem `Glyphs` Layoutobjekt. Auf die genaue strukturelle Anordnung der in der Formatierung verwendeten Layoutobjekte und eventuelle weitere nötige Bestandteile wird erst im nächsten Abschnitt ausführlich eingegangen. Offensichtlich ist aber, dass das `Glyphs` Layoutobjekt ein Kind des `Horizontal` Layoutobjektes ist (siehe hierzu auch Abbildung 4.5).
- Die vierte Formatierung, die das Layout für die Liste insgesamt beschreibt, besteht aus nur einem `Vertical` Layoutobjekt.

Selbstverständlich sind in allen Formatierungen die Parameter für die Attribute der Layoutobjekte geeignet zu setzen, um das prinzipielle Layout, das zwar bereits von den Layoutobjekten bestimmt wird, dies jedoch nur vom grundlegenden Charakter her, auch noch im Detail festzulegen. Die Attribute der Layoutobjekte werden gemäß folgender Definition Layoutattribute genannt, um sie eindeutig von den logischen Attributen abzusetzen:

Definition 42 (Layoutattribute) *Die zur Feinsteuerung der Layoutobjekte verwendeten Attribute sind eine Spezialisierung der Spezifikationsattribute und heißen Layoutattribute. Sie haben innerhalb eines Layoutobjektes einen eindeutigen Namen, über den sie angesprochen werden können. Layoutattribute sind streng typisiert und können als Werte gemäß ihrer eigenen Sorte beliebige Elemente der Sorten* `Bool`, `Number`, `Length` *oder* `String` *annehmen.*

Zur Setzung von Werten für die Layoutattribute der in den Formatierungen vorkommenden Layoutobjekte können programmiersprachliche Ausdrücke, wie sie in 4.9 eingeführt werden, verwendet werden. Die Werte der Ausdrücke müssen jedoch von der Sorte her zu den Layoutattributen, denen sie zugewiesen werden, *verträglich*[23] sein.

[23] Zur Verträglichkeit und dem dabei verwendeten *Casting*-Mechanismus siehe 4.9.5.

Layoutobjekte in [em:]-Formatierungen, die hierarchisch angeordnet sein können, sind linearisiert mit Hilfe einer Klammerstruktur beschreibbar. Diese Klammerstruktur ist schematisch wie folgt aufgebaut (die öffnende und die schließende Klammer müssen bei der Beschreibung angegeben werden, textuelle Angaben in Fettschrift müssen durch konkrete Werte ersetzt werden, andere je nach Bedarf eingefügt werden):

```
.
.
.
( Layoutobjekt-Name
 | Layoutattributliste
 | Inhalt des Layoutobjektes
) Layoutobjekt-Name
.
.
.
```

Eine Formatierung muss nicht notwendigerweise baumförmig mit einer Wurzel aufgebaut sein. Sie kann auch in Form einer Liste von (evtl. verschachtelten) Layoutobjekten vorliegen. In diesem Fall wird jedoch zur Vereinfachung der Betrachtungen angenommen, dass die Formatierung von einem *Null-Layoutobjekt*, das keine Funktionalität hat, zu einem Baum zusammengefaßt wird.

Die *Layoutattributliste* setzt sich aus einzelnen Zeilen zusammen, die jeweils die Wertsetzung für ein einzelnes Layoutattribut beinhalten. Realisiert werden die Wertsetzungen durch Tupel, die zunächst den Namen des Layoutattributes beinhalten, der, getrennt durch ein Zuweisungszeichen ($:=$), von dem zu setzenden Wert gefolgt wird. Der *Inhalt des Layoutobjektes* sind weitere Layoutobjekte, die wiederum verschachtelt sein können.

Konkrete vollständige Beispiele für Formatierungen werden im nächsten Abschnitt vorgestellt. Es ist an dieser Stelle darauf hinzuweisen, dass anstelle eines Layoutobjektes jedes beliebige im Folgenden noch eingeführte Spezifikationsobjekt stehen kann. Aus Gründen der Übersichtlichkeit wurde auf eine explizite Verdeutlichung dieser Tatsache jedoch in obigen Ausführungen verzichtet.

4.4.3 Designregeln und ihre Anwendung

Im vorigen Abschnitt wurde von Formatierungen und ihrer Aufgabe, der Beschreibung der Auswirkungen eines logischen Objektes auf das Layout eines konkreten Dokumentes, gesprochen. Weiterhin wurde angedeutet, dass mehrere Formatierungen zu einer Designstruktur für ein konkretes Dokument zusammengesetzt werden können. In diesem Abschnitt wird nun behandelt, wie dies ausgehend von einem logisch ausgezeichneten Dokument automatisch mit Hilfe von sogenannten *Designregeln* erfolgen kann.

Definition 43 (Designregel) *Eine Designregel ist eine Vorschrift, mit deren Hilfe einem logischen Objekt bei Erfüllung eines durch sie spezifizierten booleschen Prädikates eine Formatierung zugewiesen werden kann.*

Die Aufgabe einer Designregel in der Designspezifikationsmethodik [em:] an sich ist vollkommen konform zu den in Kapitel 2 durchgeführten Untersuchungen. Hervorzuheben sind jedoch die Möglichkeiten, mit denen in der Designspezifikationsmethodik [em:] die booleschen Prädikate zur Steuerung der Anwendbarkeit spezifiziert werden können. Ihnen ist ein eigener Abschnitt später in dieser Arbeit gewidmet.

Eine zusammengehörige Menge von Designregeln bildet in der Designspezifikationsmethodik [em:] im Sinne von Definition 27 eine Designspezifikation.

Die booleschen Prädikate der Designregeln werden in der Designspezifikationsmethodik [em:] auch *Umgebungsprädikate* genannt, da die in ihnen verwendbaren Kriterien Eigenschaften sind, die sich aus der Umgebung der logischen Objekte, für die Designregeln aufgestellt werden, herrühren dürfen.

Definition 44 (Umgebung eines logischen Objektes) *Die Umgebung eines logischen Objektes setzt sich in [em:] zunächst aus den für das logische Objekt gesetzten Werten der im logischen Fluss verfügbaren logischen Attribute und weiterhin der gesamten logischen Struktur des Dokumentes, in dem es enthalten ist, zusammen. Zu dieser Menge kommen noch Nebenbedingungen hinzu, die sich aus der realen Umgebung ergeben, in der das logische Objekt verarbeitet wird* [24].

Definition 45 (Umgebungsprädikat) *Die booleschen Prädikate der Designregeln werden auch Umgebungsprädikate genannt. Die zur Aufstellung der Umgebungsprädikate verwendbaren Kriterien sind Eigenschaften, die sich aus der Umgebung eines logischen Objektes zusammensetzen.*

Umgebungsprädikate in Designregeln werden in der Designspezifikationsmethodik [em:] benötigt, um z.B. logische Objekte gleicher Art (z.B. alle Paragraphen) in verschiedenen Situationen (z.B. erster Paragraph nach einer Überschrift bzw. alle anderen) unterschiedlich behandeln zu können. Aber auch Aspekte wie die benutzerangepasste Verarbeitung von logisch ausgezeichneten Dokumenten können mit der Hilfe von Umgebungsprädikaten realisiert werden. Das genaue Modell der Umgebungsprädikate, die in [em:] verwendet werden können, wird in Bezug auf die logische Struktur in Kapitel 5 beschrieben.

Zur Auffindung der Designregel, die auf ein logisches Objekt angewendet werden soll, muss ein *Matching-Algorithmus* verwendet werden. Dieser prüft zum einen, ob die in dem Umgebungsprädikat für eine Designregel angegebenen Bedingungen für das logische Objekt erfüllt sind. Zum anderen muss er bei mehreren *matchenden* Designregeln entscheiden, welche der Designregeln gewählt wird.

[24] Dies sind z.B. Informationen über die aktuelle Uhrzeit oder den Nutzer eines Dokumentes, für den das Dokument aufbereitet wird. Sie werden über logische Attribute des Wurzelknotens eines logisch ausgezeichneten Dokumentes modelliert.

Definition 46 (Matching-Algorithmus) *Ein Matching-Algorithmus ist ein Verfahren, das aufgrund einer gegebenen Menge von Designregeln für ein logisch ausgezeichnetes Dokument prüft und eindeutig reproduzierbar entscheidet, welche der verfügbaren Designregeln auf das logische Objekt angewendet wird.*

Die Forderung der Eindeutigkeit bei der Entscheidung, welche Designregel aus einer Menge der zur Verfügung stehenden Designregeln ausgewählt wird, ist von entscheidender Bedeutung für die Determiniertheit des Designspezifikationssystems [es:]. Andernfalls könnten bei zweimaliger Anwendung einer Designspezifikation auf das gleiche Dokument bei identischen Nebenbedingungen zwei verschiedene Ergebnisse entstehen – ein von den Nutzern nur schwer nachvollziehbarer und unerwünschter Aspekt.

Eine Designregel, die einem logischen Objekt `obj` bei Erfüllung des Umgebungsprädikates `p` eine Formatierung `f`, die nur aus einem Layoutobjekt `lo` besteht, zuordnet, wird in [em:] wie folgt formuliert:

$$\begin{array}{lll} \text{obj} & \leftarrow_p & \text{(lo} \\ & & \text{)lo} \end{array}$$

Wird das Umgebungsprädikat beim Zuweisungspfeil weggelassen, so handelt es sich um eine *Default-Designregel*. Ihr Umgebungsprädikat ist leer und liefert im Falle einer Auswertung immer `true` als Ergebnis.

Ein Matching-Algorithmus ermöglicht zunächst die Steuerung der lokalen Anwendung von Designregeln auf die einzelnen logischen Objekte gemäß den Vorgaben der Umgebungsprädikate. Durch die Anwendung einer Designregel auf die einzelnen logischen Objekte wird diesen eine Formatierung und somit jeweils ein Baum von Layoutobjekten zugewiesen. Man erhält *isolierte* Formatierungen für logische Objekte, die nicht miteinander verknüpft sind. Es stellt sich die Frage, über welchen Mechanismus aus diesen isolierten Formatierungen eine Designstruktur erstellt werden kann, die das Layout für das gesamte Dokument beschreibt.

Der prinzipielle Mechanismus für die Assemblierung der einzelnen Formatierungen zu einer Designstruktur ist in einfachen *Standardfällen*[25] deren hierarchische Zusammensetzung gemäß der Struktur ihrer logischen Objekte im logisch ausgezeichneten Dokument. Da eine Formatierung immer ein eindeutiges[26] Wurzelobjekt hat, ist eindeutig festgelegt, dass mit einer von der Wurzel einer Formatierung nach oben abgehenden Kante diese Formatierung auf eindeutige Art und Weise zum Anschluss an eine weiter oben liegende Formatierung verwendet werden kann.

Offen ist aber die Frage, wo diese von der unteren Formatierung eindeutig abgehende Kante an eine sie verwendende obere Formatierung angeschlossen werden soll. Abbildung 4.8 soll diese Problemstellung anhand zweier Formatierungen des Listenbeispiels aus Abschnitt 4.4.1 verdeutlichen: In dem Fall, bei dem die obere Formatierung nur aus einem Layoutobjekt besteht

[25] Als *Standardfall* wird die Situation betrachtet, in der die durch das abstrakte Dokument vorgegebene sequentielle Abfolge der logischen Objekte (lineare Struktur) im konkreten Dokument wiedergespiegelt wird.

[26] Siehe Festlegung auf Seite 110, die gegebenenfalls die Einführung eines NULL-Layoutobjektes vorschreibt.

(z.B. Formatierung für den logischen Paragraphen der Liste), ist die Frage nach dem Anschlußpunkt eindeutig klärbar; die untere Formatierung wird als Kind des einzigen Layoutobjektes der oberen Formatierung angehängt (Abbildung 4.8 a). In dem Fall, dass die obere Formatierung mehrere Blätter hat (z.B. Formatierung für den Listeneintrag), ist der Anschlußpunkt nicht mehr eindeutig festgelegt; er kann an jedem der Blätter der oberen Formatierung sein (Abbildung 4.8 b).

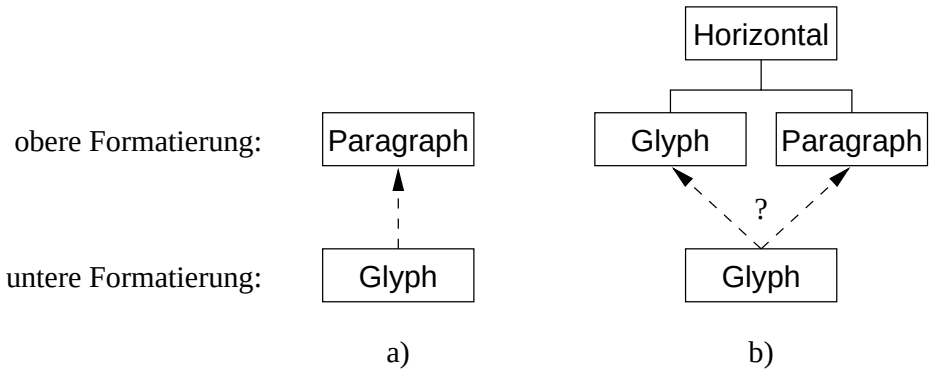

Abbildung 4.8: Mögliche Anschlußpunkte für Formatierungen

Die in der Designspezifikationsmethodik [em:] gewählte Lösung für das Problem der nicht eindeutigen Anschlusspunkte sind die sogenannten *Konnektoren*: Sie legen für eine Formatierung eines logischen Objektes `o` eindeutig die Anschlußpunkte fest, an denen untere Formatierungen an sie angeschlossen werden. Zugleich wählen sie aber auch aus, ob, und falls ja, in welcher Reihenfolge[27] die Formatierungen der logischen Kindobjekte von `o` verwendet werden.

Definition 47 (Konnektoren) *Die Klasse der Konnektoren ist eine von der Klasse der Spezifikationsobjekte abgeleitete Spezialisierung. Konnektoren werden in einer Formatierung für ein logisches Objekt* `o` *zum einen dazu verwendet, die Anschlußpunkte für Kindformatierungen festzulegen. Zum anderen sind Konnektoren aber auch* aktive *Objekte, die den Matching-Algorithmus für logische Kindobjekte von* `o` *überhaupt erst anstoßen. Dabei legen sie auch fest, ob dies von links nach rechts oder umgekehrt von rechts nach links geschieht.*

Mit der Einführung der `Konnektoren` muss nun das Beispiel der Formatierungen zu einer Liste auf Seite 109 mit Ausnahme der ersten Formatierung (für das logische Objekt `text`), die unverändert bleiben kann, ergänzt werden. Die zweite Formatierung erhält als Kind des `Paragraph` Layoutobjektes einen `Konnektor`, die dritte Formatierung als Kind des `Horizontal` Layoutobjektes und die vierte als Kind des `Vertical` Layoutobjektes. In [em:]-Notation sehen die Designregeln unter Vernachlässigung der Layoutattributlisten wie in Abbildung 4.9 dargestellt aus.

Nach Einführung der `Konnektoren` kann nun der gesamte Mechanismus verdeutlicht werden, der aufgrund eines gegebenen logisch ausgezeichneten Dokumentes und einer Designspezifikation eine Designstruktur erzeugt: Bei der Berechnung einer Designstruktur für

[27] Möglich sind die Reihenfolgen von links nach rechts und umgekehrt.

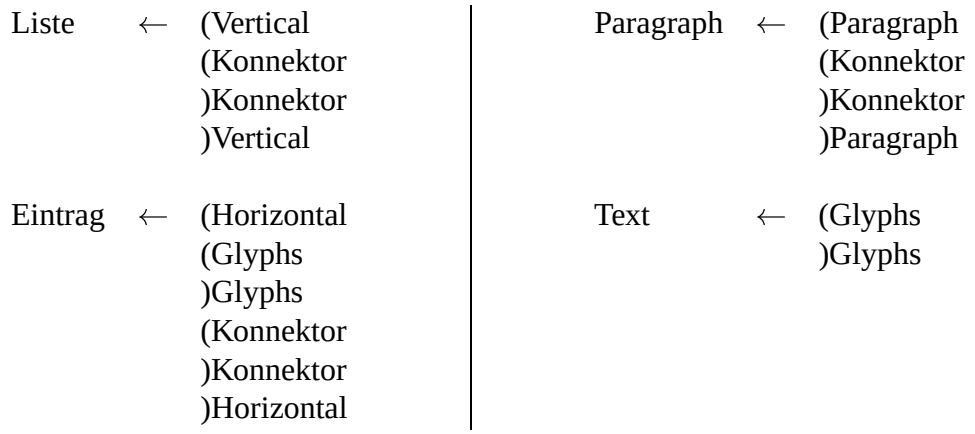

Abbildung 4.9: Beispiel einer Designspezifikation für die Liste aus Abbildung 4.7

ein logisch ausgezeichnetes Dokument wird zunächst ein Matching-Algorithmus für das logische Wurzelobjekt des logisch ausgezeichneten Dokumentes angestoßen. Die aufgrund des Matching-Algorithmus ausgewählte Formatierung wird als Wurzel für die Designstruktur verwendet und stößt von da an rekursiv mit Hilfe etwaiger in ihr enthaltener `Konnektoren` weitere Matching-Algorithmen für logische Kindobjekte an. Die dabei anzuwendenden Formatierungen werden ebenfalls in die Designstruktur aufgenommen und setzen diesen rekursiven Prozess fort.

Die Aufnahme einer Formatierung in eine Designstruktur, die auch *Einkopieren* genannt wird, erfolgt in der Regel durch strukturell identisches Einfügen einer Formatierung an die gewünschte Stelle in einer Designstruktur. Das hierfür in [em:] für normale Spezifikationsobjekte verwendete Modell ist wie folgt festgelegt: Ein in einer Formatierung vorkommendes Spezifikationsobjekt fügt die in seiner Formatierung spezifizierten Kinder sequentiell in die Designstruktur ein. Es ergibt sich damit insgesamt ein rekursives Vorgehen beim Einkopieren einer Formatierung.

Dieses soeben vorgestellte Verhalten beim Einkopieren ist jedoch nicht für alle in [em:] eingeführten Spezifikationsobjekte gültig. Bei den in [em:] vorkommenden Ausnahmeobjekten wird ein abweichendes Verhalten in dieser Arbeit aber immer explizit erklärt.

Eine Designstruktur wird somit nicht zentral gesteuert aufgrund eines vorbestimmten Algorithmus berechnet, der lediglich die logische Struktur eines gegebenen Dokumentes in vorbestimmter Weise durchläuft. Es findet vielmehr ein rekursiver, verteilt gesteuerter Aufbau der Designstruktur statt, der gemeinsam von den `Konnektoren` und dem verwendeten Matching-Algorithmus geleitet wird.

Der Mechanismus der Anwendung einer Designspezifikation auf ein logisch ausgezeichnetes Dokument mit Hilfe eines Matching-Algorithmus wird *Designapplikation* genannt:

Definition 48 (Designapplikation) *Eine Designapplikation ist eine Anwendung einer Designspezifikation auf ein logisch ausgezeichnetes Dokument. Sie erzeugt als Ergebnis eine Designstruktur. Die Anwendung der Designspezifikation auf ein logisch ausgezeichnetes Dokument*

ist ein verteilt gesteuerter Prozess. Er wird angestoßen, indem auf das logische Wurzelobjekt des gegebenen logisch ausgezeichneten Dokumentes ein Matching-Algorithmus angewendet wird. Von da an steuern die in den Formatierungen der gewählten Designregeln vorkommenden `Konnektoren` *im Zusammenspiel mit dem Matching-Algorithmus den weiteren Aufbau der Designstruktur.*

Nach der Applikation einer Designspezifikation auf ein logisch ausgezeichnetes Dokument sind nicht mehr alle der in der zum Aufbau der Designstruktur angewendeten Formatierungen vorkommenden Spezifikationsobjekte in der erzeugten Designstruktur vorhanden. Die neben der Klasse der Layoutobjekte bisher noch eingeführte Klasse der `Konnektoren` hat in [em:] nur innerhalb einer Designregel eine Funkionalität. Innerhalb einer Designstruktur sind Instanzen der Klasse `Konnektor` *überflüssig*, sie tragen nichts mehr zur Layoutspezifikation bei. Aus diesem Grund werden sie beim Aufbau einer Designstruktur beim Einkopieren ihrer Formatierung einfach weggelassen. Mit ihnen verbundene Kindobjekte (insbesondere die Formatierungen, die an sie angeschlossen werden) werden in geordneter Folge an das jeweilige Vaterobjekt der `Konnektoren` angeschlossen.

In Abbildung 4.10 wird ausführlich der schrittweise Aufbau einer Designstruktur aufgrund der in Abbildung 4.7 gegebenen logisch ausgezeichneten Liste und der in Abbildung 4.9 vorgegebenen Designspezifikation dargestellt.

Im ersten Schritt wird aufgrund des initialen Aufrufes des Matching-Algorithmus auf das logische Objekt `Liste` die erste Designregel aus Abbildung 4.9 angewendet und als Wurzel in die zu erstellende Designstruktur einkopiert (Abbildung 4.10 a). Der in der Designregel enthaltene `Konnektor` wird dabei ausgespart; er initiiert jedoch einen weiteren Aufruf des Matching-Algorithmus für alle logischen Kindobjekte der `Liste`. Aus diesem Grund wird für den ersten `Eintrag` der `Liste` die entsprechende `Listen`-Formatierung in die Designstruktur (Abbildung 4.10 b) einkopiert. Der wiederum ausgesparte `Konnektor` ruft nun den Matching-Algorithmus für das logische Objekt `Paragraph` auf, wodurch ein Layoutobjekt `Paragraph` in die Designstruktur eingefügt (Abbildung 4.10 c) und ein weiteres Mal der Matching-Algorithmus, diesmal für das logische Objekt `Text`, aufgerufen wird. Letzterer Aufruf fügt ein `Glyphs` Layoutobjekt in die Designstruktur ein (Abbildung 4.10 d) und beendet die Verarbeitung des ersten logischen `Eintrag`es. Das zweite logische Objekt `Eintrag` wird analog zum Vorgehen beim ersten `Eintrag` abgearbeitet. Es ergibt sich insgesamt die in Abbildung 4.10 e dargestellte Designstruktur.

4.4.4 Der Applikationspfad

Im Verlauf einer Designapplikation entsteht ein *Applikationspfad* durch das logische Dokument. Er ergibt sich aufgrund der beim Zusammenspiel des Matching-Algorithmus und der `Konnektoren` besuchten logischen Objekte des Ausgangsdokumentes. Der Applikationspfad durchläuft die logischen Objekte in der sequentiellen Reihenfolge, wie versucht wird, Designregeln

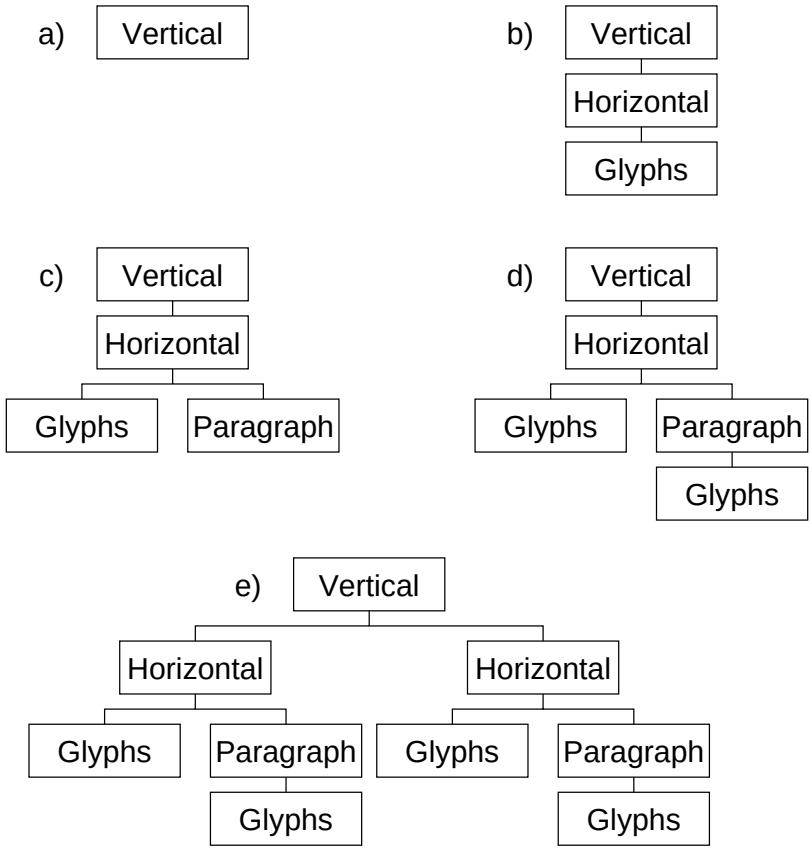

Abbildung 4.10: Schrittweiser Aufbau einer Designstruktur

auf die logischen Objekte anzuwenden[28]. Ein Applikationspfad wird in der Regel *sehr ähnlich* zu der linearisierten Struktur eines logisch ausgezeichneten Dokumentes sein, kann aber bei geeignetem Aufbau der in einer Designspezifikation enthaltenen Designregeln auch *wirr* durch das logische Dokument verlaufen.

Definition 49 (Applikationspfad) *Der Applikationspfad einer Designspezifikation ist die sequentielle Abfolge der logischen Objekte, auf die eine Designregel angewendet wurde, gemäß der Reihenfolge, in der versucht wurde, auf sie während einer Designapplikation eine Designregel anzuwenden. Diese Reihenfolge ergibt sich aufgrund der Zusammenarbeit zwischen den in den Formatierungen vorkommenden* `Konnektoren` *und dem verwendeten Matching-Algorithmus.*

Folgt man dem beim Verlauf einer Designapplikation entstehenden Applikationspfad, so kann es passieren, dass man dabei mehrfach ein und dasselbe logische Objekt passiert, während andere, ebenfalls im verarbeiteten Dokument vorkommende logische Objekte, nicht traversiert

[28] Nicht bei jedem Aufruf eines Matching-Algorithmus auf ein logisches Objekt muss eine passende Designregel gefunden werden; es ist möglich und sinnvoll, für einige logische Objekte überhaupt keine Designregel in einer Designspezifikation bereitzustellen. In diesem Fall wird dann auch keines der logischen Kindobjekte verarbeitet.

werden. Dies bedeutet, dass einige logische Objekte mehrfach zum Layout eines Dokumentes beitragen können, während andere logische Objekte überhaupt nichts dazu beitragen.

Im Folgenden wird die Bedeutung der Applikationspfade anhand eines kleinen logisch ausgezeichneten Dokumentes (siehe Abbildung 4.11) aufgezeigt und daran seine für die Designspezifikationsmethodik [em:] verwendbaren Auswirkungen erklärt. Dabei wird die in Abbildung 4.12 dargestellte Designspezifikation zugrundegelgt.

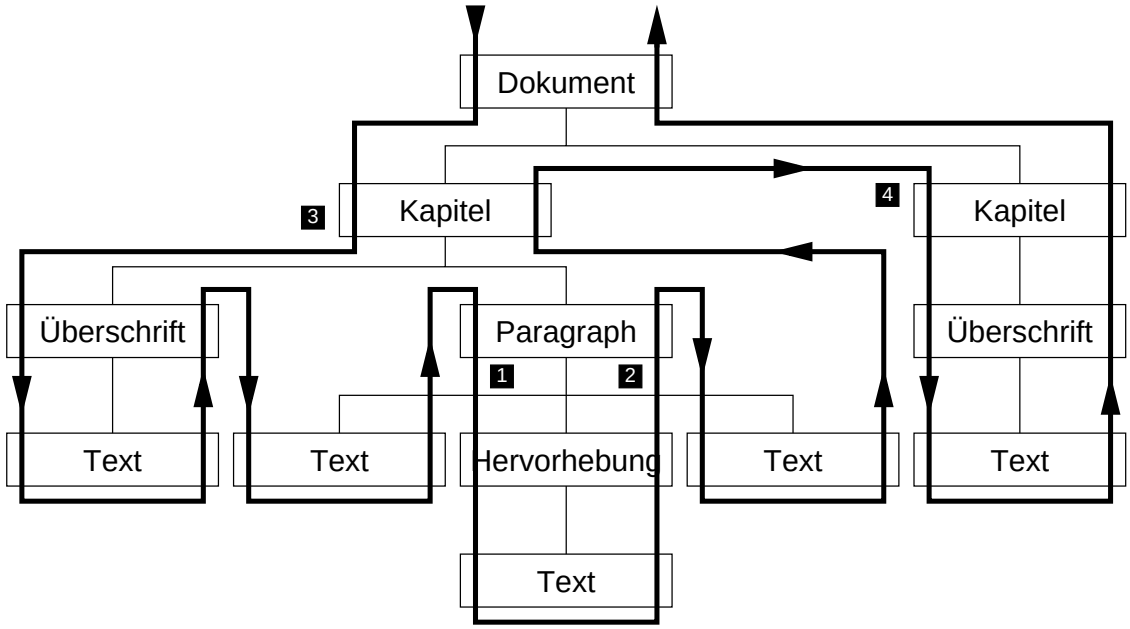

Abbildung 4.11: Logisch ausgezeichnetes Beispieldokument mit einem Applikationspfad: Der fett eingezeichnete Applikationspfad verläuft gemäß der in 4.12 vorgegebenen Designspezifikation in einem *depth-first*-Pfad durch das gegebene logisch ausgezeichnete Dokument. Schneidet er links ein logisches Objekt, so bedeutet dies, dass auf das logische Objekt eine Designregel angewendet und danach mit der Bearbeitung der logischen Kindobjekte begonnen wird. Schneidet er rechts ein logisches Objekt, symbolisiert dies die Beendigung der Verarbeitung der logischen Kindobjekte und den Abschluß der Designregel-Anwendung auf das logische Objekt. Von Position 1 bis Position 2 auf dem Applikationspfad gilt die Layouteigenschaft, dass Text in 12pt gesetzt wird. Ab Position 3 wird für die Kapitelnummerierung die Ziffer „1“ verwendet, ab Position 4 die Ziffer „2“. Näheres hierzu siehe im Fließtext.

Bedeutung des Applikationspfades

Ein Applikationspfad führt in der Regel an den logischen Objekten eines zu layoutenden Dokumentes in der Reihenfolge entlang, in der sie im konkreten Dokument erscheinen sollen[29]. Folgt

[29]Ausnahmen hiervon bilden sogenannte *gleitende* Layoutbestandteile wie z.B. Bilder, die bei geeigneten Seitenumbruchs-Algorithmen zwar *synchronisiert* von ihrer Umgebung, aber lokal losgelöst von ihr, angeordnet werden können.

Dokument	←	(Window (Vertical (Konnektor)Konnektor)Vertical)Window	Kapitel	←	(Konnektor)Konnektor
Überschrift	←	(Paragraph (Konnektor)Konnektor)Paragraph	Paragraph	←	(Paragraph (Konnektor)Konnektor)Paragraph
Text	←	(Glyphs)Glyphs	Hervorhebung	←	(Konnektor)Konnektor

Abbildung 4.12: Beispiel einer Designspezifikation für das Dokument aus Abbildung 4.11

man diesem Pfad gedanklich, so gelten ab bestimmten Punkten (linkes Schneiden eines logischen Objektes) zumindest für eine besimmte Zeit lang (in der Regel solange, wie die Kinder des Objektes verarbeitet werden; bis zum rechten Schneiden des logischen Objektes) gewisse Eigenschaften für das zu erstellende Layout der folgenden logischen Objekte.

So ist es z.B. üblich, dass in einem den allgemein gültigen Vorgaben entsprechenden Diplomarbeitstext die Schriftgröße innerhalb normaler Paragraphen 12pt beträgt, egal ob es sich um normalen oder hervorgehobenen Text handelt. In Abbildung 4.11 gilt diese Eigenschaft der Schriftgröße von 12pt auf dem Applikationspfad zumindest vom linken (Markierung 1 in Abbildung 4.11) bis zum rechten (Markierung 2 in Abbildung 4.11) Entlanglaufen am logischen Objekt `Paragraph`. Eine weitere Layouteigenschaft, die entlang des Applikationspfades gilt, ist die Tatsache, dass ab dem ersten logischen Objekt `Kapitel` (Markierung 3 in Abbildung 4.11) bei der Nummerierung für Überschriften eine „1“ und ab dem zweiten logischen Objekt `Kapitel` (Markierung 4 in Abbildung 4.11) eine „2“ für die Kapitelebene verwendet werden muss. Diese Eigenschaft, dass der Applikationspfad an bestimmten Stellen gewisse zu setzende Werte für die Layoutobjekte besitzt, wird in [em:] vereinfacht ausgedrückt mit Hilfe eines den Applikationspfad entlanglaufenden *Spezifikationsvariablenflusses* nachgebildet und zur Setzung der Werte von Spezifikationsattributen benutzt.

4.4.5 Der Spezifikationsvariablenfluss

Es würde einen erheblichen Aufwand bedeuten, sämtliche Spezifikationsattribute aller Spezifikationsobjekte in einer Designspezifikation bzw. Designstruktur von Hand explizit mit einem Wert belegen zu müssen. Dies ist insbesondere auch nicht immer sinnvoll, da für Designregeln evtl. erst bei ihrem Einkopieren in eine Designstruktur aufgrund der Anwendungsstelle im

logisch ausgezeichneten Dokument klar ist, welche konkreten Werte für die Spezifikationsattribute gesetzt werden müssen. Wie oben aufgezeigt wurde ist es außerdem wünschenswert, Spezifikationsattribute von verschiedenen Spezifikationsobjekten ohne großen Aufwand mit dem gleichen Wert zu versehen (z.B. soll die Schrift innerhalb einer Hervorhebung die gleiche Größe haben wie im übrigen umgebenden Text auch, selbst wenn die Schriftart für die Hervorhebung geändert wurde).

Die prinzipielle Lösung dieser Aufgabe übernehmen in [em:] die *Spezifikationsvariablen*, die zur Wertsetzung von Spezifikationsattributen beim Einkopieren von Spezifikationsobjekten in eine Designstruktur verwendet werden können. Für jedes Spezifikationsattribut, das von einem Spezifikationsobjekt in [em:] bereitgestellt wird, wird in [em:] eine Spezifikationsvariable gleichen Namens und gleicher Sorte automatisch eingeführt und mit einem von den Nutzern vorgebbaren *Default-Wert* besetzt. Falls nun ein Spezifikationsobjekt bei einer Anwendung einer Regel ein Spezifikationsattribut enthält, für das kein expliziter Wert gesetzt wurde, wird stattdessen *automatisch* der aktuelle Wert der Spezifikationsvariablen gleichen Namens eingesetzt. Die automatische Einführung einer gleichnamigen Spezifikationsvariablen für jedes in [em:] vorkommende Spezifikationsattribut führt dabei eine gewisse Fehlertoleranz ein: Es kann unter jeder Bedingung für alle Spezifikationsattribute ein Wert gesetzt werden.

Definition 50 (Spezifikationsvariable) *In [em:] wird für jedes deklarierte Spezifikationsattribut eine Spezifikationsvariable gleichen Namens und gleicher Sorte automatisch eingeführt*[30]*. Für jede Spezifikationsvariable wird bei ihrer Einführung ein Default-Wert gesetzt, der von den Nutzern gezielt vorgegeben werden kann. Der aktuelle Wert von Spezifikationsvariablen wird bei der Anwendung einer Designregel für diejenigen Spezifikationsattribute gleichen Namens eingesetzt, für die kein expliziter Ausdruck vorgegeben ist. Weiterhin können von den Nutzern der Designspezifikationsmethodik [em:] Spezifikationsvariablen auch frei eingeführt werden, um aufgabenspezifisch Layouteigenschaften berechnen zu können. Spezifikationsvariablen sind wie Spezifikationsattribute streng typisiert und können als Werte gemäß ihrer eigenen Sorte beliebige Elemente der Sorten* `Bool`, `Number`, `Length` *oder* `String` *annehmen.*

Schematisch betrachtet fließen die Spezifikationsvariablen in [em:] in einer zu den logischen Attributen ähnlichen Weise durch eine Baumstruktur, der *erweiterten Designstruktur*. Diese erweiterte Designstruktur beinhaltet zusätzlich zu den in einer Designstruktur einkopierten Formatierungen auch noch die logischen Objekte, für die die Formatierungen verwendet wurden. Sie werden oberhalb ihrer Formatierungen einkopiert. Weiterhin enthält die erweiterte Designstruktur Spezifikationsobjekte, die in einer *normalen* Spezifikationsstruktur keine Funktion mehr haben. Sie können im Rahmen dieses Flusses dazu verwendet werden, die Werte von Layoutattributen zu setzen: Wie schon in Abbildung 4.11 (unterschiedliche Wertsetzung für die Schriftgröße) gezeigt wurde, ist es nämlich nötig, im Verlauf des Spezifikationsvariablenflusses den Wert von Spezifikationsvariablen zu verändern, um spezifische Layouteigenschaften im Verlauf einer Designapplikation zielgerecht steuern zu können. Dies geschieht in [em:] im Wesentlichen mit Hilfe von zwei neuen Spezifikationsobjekten, dem `Inheritance` und dem

[30] Dies bedingt indirekt, dass in [em:] keine zwei Spezifikationsvariablen mit gleichem Namen, aber unterschiedlicher Sorte eingeführt werden dürfen (siehe Definition 39, Spezifikationsobjekte, auf Seite 108).

`Modifier` Spezifikationsobjekt, die beide in Formatierungen zur Anwendung kommen. Bevor nun auf die Details dieser und weiterer neuer Spezifikationsobjekte eingegangen wird, werden zunächst einige Begriffe eingeführt:

Definition 51 (Erweiterte Designstruktur) *Erweiterte Designstrukturen in [em:] sind Hilfskonstrukte, die der Veranschaulichung von Berechnungsvorgängen für die Wertsetzung von Spezifikationsattributen beim Aufbau einer Designstruktur dienen. Die erweiterte Designstruktur entsteht deshalb wie eine* normale *Designstruktur. In Ergänzung zu einfachen Designstrukturen enthält sie jedoch oberhalb aller Formatierungen das logische Objekt, für das die Formatierungen verwendet werden. Ausserden werden mit Ausnahme der* `Konnektoren` *alle in den Formatierungen vorkommenden Spezifikationsobjekte in die erweiterte Designstruktur einkopiert.*

Ein Tiefendurchlauf durch eine erweiterte Designstruktur enthält somit die logischen Objekte in genau der Reihenfolge, in der sie auch im Applikationspfad enthalten sind. Der Übergang einer erweiterten Designstruktur zu einer Designstruktur erfolgt einfach, indem die nicht zulässigen Spezifikationsobjekte (dies sind Spezifikationsobjekte, die in einer Designstruktur keine Funktionalität mehr haben) sowie die logischen Objekte aus der erweiterten Designstruktur entfernt werden.

Definition 52 (Spezifikationsvariablenfluss) *Der Spezifikationsvariablenfluss in [em:] stellt symbolisch die Verfügbarkeit von Spezifikationsvariablen und der durch sie eingebrachten Werte in einer erweiterten Designstruktur dar. Er fließt dazu strukturell betrachtet in einem depth-first Durchlauf an allen in einer erweiterten Designstruktur enthaltenen Objekten entlang. Im Laufe des Spezifikationsvariablenflusses können die Werte von einzelnen Spezifikationsvariablen mit Hilfe von ausgewählten Spezifikationsobjekten (den Auswertungsobjekten) manipuliert werden.*

Nomenklatur: Um die Berührungspunkte zwischen dem Spezifikationsvariablenfluss und den Spezifikationsobjekten in einer erweiterten Designstruktur immer klar und eindeutig bezeichnen zu können, wird folgende Nomenklatur festgelegt. Diese wird später bei der Erklärung einzelner Spezifikationsobjekte verwendet.

Betreten	Erstes Eintreten des Spezifikationsvariablenflusses in das Spezifikationsobjekt.
Fortfahren	Verlassen des Spezifikationsobjektes durch den Spezifikationsvariablenfluss in Richtung Unterbaum.
Rückkehren	Wiedereintritt des Spezifikationsvariablenflusses in das Spezifikationsobjekt nach Umfließen des Unterbaumes.
Verlassen	Endgültiges Verlassen des Spezifikationsobjektes nach rechts (in Richtung Vater oder Bruder) durch den Spezifikationsvariablenfluss.

Berühren Eintreten des Spezifikationsvariablenflusses in das Spezifikationsobjekt und zugleich unverändertes `Vorbeifließen` am Spezifikationsobjekt.

Vorbeifließen Kontaktloses Entlangfließen des Spezifikationsvariablenflusses am Spezifikationsobjekt.

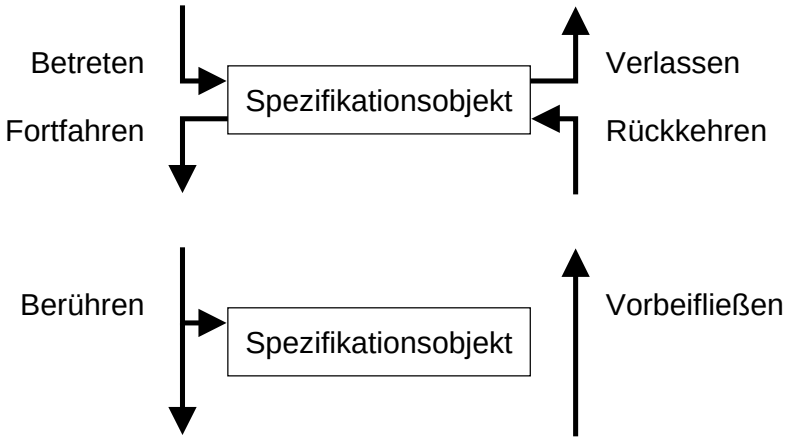

Abbildung 4.13: Nomenklatur für den Spezifikationsvariablenfluss

Manipulationsmöglichkeiten am Spezifikationsvariablenfluss

Im Folgenden werden der Spezifikationsvariablenfluss und die Möglichkeiten zu dessen Manipulation sowie dessen Auswirkungen auf die Wertsetzung der Spezifikationsattribute von Spezifikationsobjekten beim Aufbau einer erweiterten Designstruktur vorgestellt. Dies erfolgt jeweils am Beispiel einer einzelnen Spezifikationsvariablen `v` von der Sorte `Number`[31]. In Ergänzung hierzu ist aber darauf hinzuweisen, dass in den folgenden Abbildungen in dem durch die Pfeile angedeuteten Spezifikationsvariablenfluss prinzipiell immer *alle* verfügbaren Spezifikationsvariablen fließen.

Layoutobjekte haben keine Möglichkeit, manipulierend in den Spezifikationsvariablenfluss einzugreifen, er fließt an ihnen unverändert vorbei. Zur Berechnung der Werte für ihre Layoutattribute findet lediglich eine `Berührung` der Layoutobjekte durch den Spezifikationsvariablenfluss im Sinne obiger Nomenklatur statt. Der aktuelle Wert einer Spezifikationsvariablen wird beim linken Berühren zur Setzung des Wertes eines Layoutattributes verwendet. Abbildung 4.14 soll dies verdeutlichen: Der Wert der Spezifikationsvariablen `v` ist im Spezifikationsvariablenfluss vor dem Berühren des Layoutobjektes 1 und wird entsprechend zur Setzung des Layoutattributes `x` des Layoutobjektes benutzt. Nach dem Umfließen des Unterbaumes kehrt der Spezifikationsvariablenfluss zurück und fließt an dem Layoutobjekt ohne eine weitere Beeinflussung vorbei. Da wir nun annehmen, dass der Wert von `v` im Spezifikationsvariablenfluss beim Umfließen des Unterbaumes nicht geändert wurde, hat `v` beim rechten Vorbeifließen am Layoutobjekt nach wie vor den Wert 1.

[31] Um darzustellen, wie unmanipulierte Spezifikationsvariablen im Spezifikationsvariablenfluss fließen, wird vereinzelt eine zweite Spezifikationsvariablen `x` (ebenfalls von der Sorte `Number`) verwendet

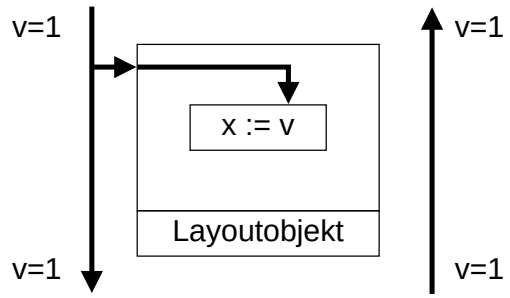

Abbildung 4.14: Spezifikationsvariablenfluss bei einem Layoutobjekt: Der Spezifikationsvariablenfluss wird in das Layoutobjekt abgezweigt und dort ausschließlich zur Setzung des Wertes eines Layoutattributes verwendet. Die innere Box im Layoutobjekt symbolisiert die Zuweisung eines Wertes an ein Layoutattribut.

Im Folgenden werden mehrere neue Spezifikationsobjekte vorgestellt, die im Gegensatz zu den oben behandelten Layoutobjekten auf verschiedene Arten manipulierend auf den Spezifikationsvariablenfluss einwirken können. Sie werden Auswertungsobjekte genannt:

Definition 53 (Auswertungsobjekte) *Auswertungsobjekte sind in der Designspezifikationsmethodik [em:] Spezifikationsobjekte, die manipulierend auf den Spezifikationsvariablenfluss einwirken können. Sie verfügen dazu gegebenenfalls über Spezifikationsattribute, über die ihr Verhalten bei der Steuerung des Spezifikationsvariablenflusses gesteuert werden kann. Auswertungsobjekte können weiterhin Listen für Spezifikationsvariablen bereitstellen, über die explizit von den Anwendern die Spezifikationsvariablen (samt der auf ihr auszuführenden Manipulation) angegeben werden können, die im Spezifikationsvariablenfluss manipuliert werden sollen.*

Nach Definition 53 können **Auswertungsobjekte** über Spezifikationsattribute verfügen. In den Fällen, da dies zutrifft, erfolgt die Wertsetzung für die Spezifikationsattribute analog zu dem in Abbildung 4.14 vorgestellten Verfahren wie bei den Layoutobjekten. Dies wird auch, falls nötig, in den folgenden Abbildungen durch in Auswertungsobjekten enthaltene weiß ausgefüllte Boxen dargestellt. Im Gegensatz dazu werden die Manipulationen, die Auswertungsobjekte auf dem Spezifikationsvariablenfluss ausführen, mit Hilfe von grau ausgefüllten Boxen visualisiert.

Das **Inheritance** Spezifikationsobjekt ist eines der Auswertungsobjekte, das den Wert einer Spezifikationsvariablen im Spezifikationsvariablenfluss während einer Designapplikation verändern kann. Es stellt hierzu eine anwenderdefinierbare Spezifikationsvariablenliste, welche sich nach außen hin wie die Layoutattributliste eines Layoutobjektes präsentiert, zur Verfügung. Das heißt, man kann den einzelnen Spezifikationsvariablen einen Ausdruck zuweisen, der wie für die Layoutattribute der Layoutobjekte berechnet wird; anschließend ist jedoch für alle im Unterbaum des `Inheritance` Auswertungsobjektes liegenden Spezifikationsobjekte unter der Spezifikationsvariablen `v` im Spezifikationsvariablenfluss nur noch dieser neu berechnete Wert zugänglich. Nach dem Umfließen des Unterbaumes erhält der Wert der Spezifikationsvariablen `v` dann durch das `Inheritance` Auswertungsobjekt wieder seinen alten Wert. Abbildung 4.15 zeigt die Wirkung des `Inheritance` Auswertungsobjektes noch einmal in einer graphischen Darstellung.

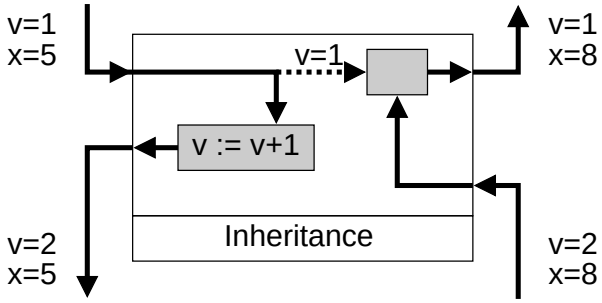

Abbildung 4.15: Spezifikationsvariablenfluss bei einem `Inheritance` Auswertungsobjekt: Das `Inheritance` Auswertungsobjekt verändert den Spezifikationsvariablenfluss gemäß den in der linken grauen Box getroffenen Vorgaben (nur die Spezifikationsvariable `v`) und schickt ihn entsprechend weiter nach unten. Beim Verlassen des `Inheritance` Auswertungsobjektes wird der Spezifikationsvariablenfluss für die Spezifikationsvariable `v` wieder auf den beim Betreten des `Inheritance` Auswertungsobjektes gültigen Wert restauriert (mit Hilfe des gestrichelten Pfeiles).

Beinhaltet die Spezifikationsvariablenliste eines `Inheritance` Auswertungsobjektes mehrere Spezifikationsvariablen, so findet semantisch eine parallele Auswertung der Zuweisungen[32] statt, wodurch die Reihenfolge der Einträge in der Spezifikationsvariablenliste innerhalb eines `Inheritance` Auswertungsobjektes keine Rolle spielt.

Formal betrachtet ist jeder Eintrag in der Spezifikationsvariablenliste eines `Inheritance` Auswertungsobjektes eine Spezifikationsvariablendeklaration

```
Sort v := E,
```

die eine neue Spezifikationsvariable `v` einführt und zugleich mit einem Wert `E` initialisiert. `Sort` kann dabei eine der gültigen Sorten für eine Spezifikationsvariable sein.

Durch eine Deklaration für eine Spezifikationsvariable `v` wird eine bereits deklarierte gleichnamige Spezifikationsvariable `v` verschattet. Die durch diese Tatsache zu verwendenden Begriffe und Auswirkungen sind dieselben, wie sie schon bei den logischen Attributen und dem logischen Fluss auf Seite 105 eingeführt wurden.

Das **Modifier** Auswertungsobjekt ist das zweite Auswertungsbjekt, welches den Wert einer Spezifikationsvariablen im Spezifikationsvariablenfluss verändern kann. Hierzu bietet es wie das `Inheritance` Auswertungsobjekt eine benutzerdefinierbare Liste von Spezifikationsvariablen an, mit deren Hilfe die zu beinflussenden Spezifikationsvariablen des Spezifikationsvariablenflusses angegeben werden können. Zugleich können die Anwender den Spezifikationsvariablen auch beim `Modifier` Auswertungsobjekt in der Liste der Spezifikationsvariablen einen Ausdruck der [em:]-Ausdruckssprache zuordnen[33].

Der Unterschied zum `Inheritance` Auswertungsobjekt hinsichtlich der Wirkung auf den Spezifikationsvariablenfluss besteht darin, dass die `Modifier` Auswertungsobjekte den Wert

[32] Innerhalb eines `Inheritance` Auswertungsobjektes ist es nicht erlaubt, mehrfach für eine Spezifikationsvariable einen Wert zu berechnen.

[33] Auch beim `Modifier` Auswertungsobjekt ist es wiederum nicht zulässig, mehrfach für eine Spezifikationsvariable einen Wert zu berechnen.

einer Spezifikationsvariablen nicht bereits beim `Fortfahren` verändern, sondern erst beim `Verlassen`. Somit ist also nicht der eigene Unterbaum der Zielbereich der Auswirkungen eines `Modifier` Auswertungsobjektes, sondern die gesamte erweiterte Designstruktur und der sich dabei ergebende Spezifikationsvariablenfluss, die nach der vollständigen Abhandlung des `Modifier` Auswertungsobjektes und dessen Unterbaum noch bearbeitet werden. Abbildung 4.16 visualisiert diese Aussage.

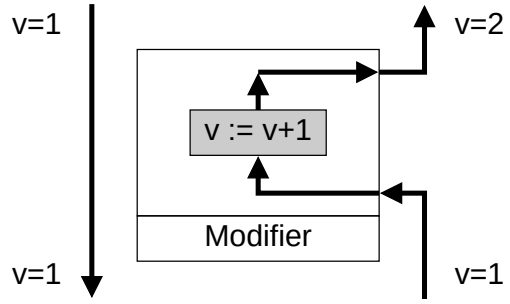

Abbildung 4.16: Spezifikationsvariablenfluss bei einem `Modifier` Auswertungsobjekt: Der Spezifikationsvariablenfluss wird erst nach dem Rückkehren, jedoch noch vor dem Fortfahren, gemäß den Vorgaben in der grauen Box verändert.

Zu beachten ist somit, dass die Auswertung der in `Modifier` Auswertungsobjekten festgelegten Zuweisungen am Spezifikationsvariablenfluss erst beim `Rückkehren` in die `Modifier` Auswertungsobjekte vorgenommen wird; ausschlaggebend für die Wertsetzung sind also die aktuellen Spezifikationsvariablenwerte aus dem *Rückfluss* in ein `Modifier` Auswertungsobjekt. Diese Eigenschaft der Manipulation des Spezifikationsvariablenflusses ist insbesondere bei einem vernesteten Vorkommen von `Modifier` Auswertungsobjekten von Bedeutung: Die in tiefer liegenden `Modifier` Auswertungsobjekten vorgenommenen Änderungen an Spezifikationsvariablen im Spezifikationsvariablenfluss, die ja beim linken `Vorbeifließen` an einem oberen `Modifier` Auswertungsobjekt noch nicht gültig waren, werden trotzdem beachtet.

Denkbar wäre beim Entwurf der `Modifier` Auswertungsobjekte auch ein Design gewesen, bei dem die Auswirkungen der `Modifier` Auswertungsobjekte bereits beim `Fortfahren` in Kraft treten würden[34]. Dieser Verhalten ist aber vermutlich für die [em:]-Zielgruppe schwieriger zu verstehen, als es beim gewählten Vorgehen der Fall ist. Ein typisches Beispiel, das dies verdeutlichen soll, ist der Einsatz des `Modifier` Auswertungsobjektes zum Zählen von z.B. Kapiteln mit Hilfe einer Spezifikationsvariablen `nr` der Sorte `Number`. Beim aktuell gewählten Ansatz kann die Spezifikationsvariable `nr` mit dem Wert `1` vorbelegt werden und bei jedem Vorkommen eines Kapitels um eins erhöht werden. Bei dem Ansatz, der die Auswirkungen der `Modifier` Auswertungsobjekte beim `Fortfahren` zur Geltung bringt, muss die

[34]Dieses Verhalten ist auch mit der derzeitigen Funktionalität des `Modifier` Auswertungsobjektes erzielbar, indem ein `Modifier` Auswertungsobjekt so in eine Designregel eingebaut wird, dass es vor dem Abarbeiten der restlichen Regel wieder verlassen wird (z.B. durch eine Verwendung als ein linker Nachbar desjenigen Spezifikationsobjektes, für das der Effekt bereits gelten soll).

Spezifikationsvariable `nr` hingegen mit `0` vorbelegt werden; eine Tatsache, die *Nichtinformatikern* nicht intuitiv zugänglich sein wird.

Wie im `Inheritance` Auswertungsobjekt werden auch im `Modifier` Auswertungsobjekt alle Zuweisungen semantisch wieder parallel vorgenommen und laufen deshalb ohne gegenseitige Beeinflussung in jeglicher Reihenfolge mit identischem Ergbnis ab.

Hervorzuheben sind die Auswirkungen, die eine hierarchische Kombination eines `Inheritance` mit einem `Modifier` Auswertungsobjekt, die beide dieselbe Spezifikationsvariable `v` verändern, haben. Zur Veranschaulichung wird hierzu Abbildung 4.17 referenziert. Beim Aufbau der erweiterten Designstruktur kommt die Spezifikationsvariable `v` beim obenliegenden `Inheritance` Auswertungsobjekt mit dem Wert 1 an und wird auf den Wert 3 gesetzt im Spezifikationsvariablenfluss weitergeschickt. Im `Modifier` Auswertungsobjekt wird der Wert der Spezifikationsvariablen `v` um eins erhöht und somit auf 4 gesetzt. Beim Rückkehren in das `Inheritance` Auswertungsobjekt wird `v` aber wieder auf den Wert 1 zurückgesetzt und derart beim Verlassen im Spezifikationsvariablenfluss weitergeschickt. Das obenliegende `Inheritance` Auswertungsobjekt *begrenzt* also die Auswirkungen des untenliegenden `Modifier` Auswertungsobjektes, die sich normalerweise auf den restlichen Spezifikationsvariablenfluss durch die erweiterte Designstruktur erstrecken würden, auf den Unterbaum des `Inheritance` Auswertungsobjektes.

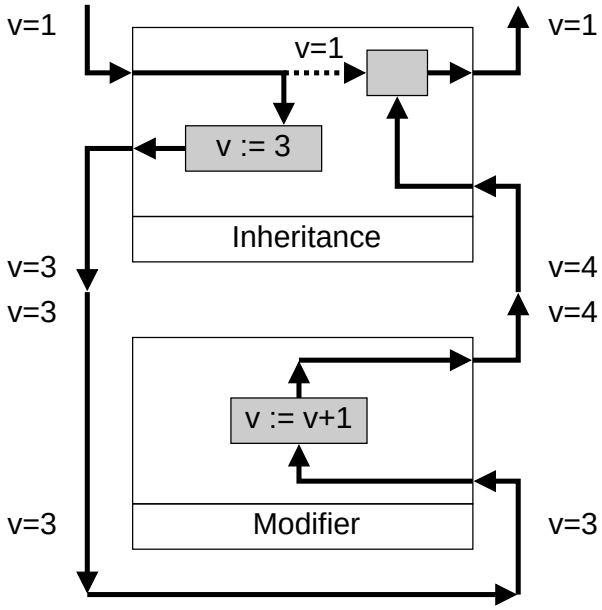

Abbildung 4.17: Kombination eines `Inheritance` und eines `Modifier` Auswertungsobjektes: Die Auswirkungen des untenliegenden `Modifier` Auswertungsobjektes auf die Spezifikationsvariable `v` werden durch das obenliegende `Inheritance` Auswertungsobjekt auf den Unterbaum des `Inheritance` Auswertungsobjektes begrenzt.

Das **Blocker** Auswertungsobjekt verallgemeinert systematisch den Effekt der `Inheritance` Auswertungsobjekte, die Auswirkungen von Spezifikationsvariablen-Änderungen auf

Teilbäume zu beschränken. Im Unterschied zu den `Inheritance` Auswertungsobjekten bietet das `Blocker` Auswertungsobjekt jedoch keine Spezifikationsvariablenliste zur Auswahl spezifischer Spezifikationsvariablen, für die die Auswirkungen eingeschränkt werden sollen, an; es werden immer für *alle* Spezifikationsvariablen im Spezifikationsvariablenfluss die Auswirkungen auf den Unterbaum des `Blocker` Auswertungsobjektes eingeschränkt. Abbildung 4.18 verdeutlicht dies wiederum graphisch am Beispiel einer Spezifikationsvariablen `v` aus dem Spezifikationsvariablenfluss.

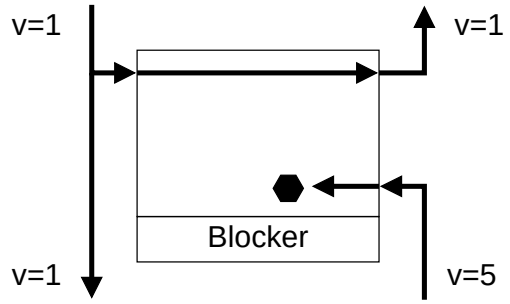

Abbildung 4.18: Spezifikationsvariablenfluss bei einem `Blocker` Auswertungsobjekt: Beim linken Berühren eines `Blocker` Auswertungsobjektes werden alle verfügbaren Spezifikationsvariablen gesichert und können deshalb beim Verlassen des Spezifikationsvariablenflusses auf diese Werte zurückgesetzt werden.

Es folgen nun noch zwei weitere Auswertungsobjekte, die den Spezifikationsvariablenfluss beeinflussen können. Im Gegensatz zu den bisher eingeführten Auswertungsobjekten dieser Klasse haben die folgenden Auswertungsobjekte aber auch noch Auswirkungen auf das Einkopieren ihrer Kindobjekte respektive auf ihr eigenes Einkopieren in die (erweiterte) Designstruktur.

Das folgend vorgestellte Auswertungsobjekt, das den Spezifikationsvariablenfluss beeinflusst, ist das **Repeater** Auswertungsobjekt. Es liest aus einer Datei (deren Name im Spezifikationsattribut `inputFileName` gesetzt wird) die Werte für dort deklarierte Spezifikationsvariablen, deklariert sie wie ein `Inheritance` Auswertungsobjekt und gibt sie im Spezifikationsvariablenfluss weiter. Die *Besonderheit* des `Repeater` Auswertungsobjektes ist dabei, dass in der betreffenden Datei mehrere Sätze von Spezifikationsvariablenwerten stehen können; der Unterbaum des `Repeater` Auswertungsobjektes wird in diesem Fall so oft ausgewertet, wie Datensätze in der Datei vorhanden sind. Das `Repeater` Auswertungsobjekt kann also das Einkopieren seines Unterbaumes in einer erweiterten Designstruktur beliebig oft, je nach den Vorgaben in seiner Eingabedatei, veranlassen[35].

Die Funktionalität des `Repeater` Auswertungsobjektes wird in Abbildung 4.19 verdeutlicht. Das `Repeater` Auswertungsobjekt selbst erscheint in keiner Designstruktur, wohl aber in der erweiterten Designstruktur. Die verschiedenen Auswertungsdurchgänge sind dann Kinder des `Repeater` Auswertungsobjektes.

[35]Ein `Repeater` Spezifikationsobjekt kann somit beispielswiese zur Erstellung von Serienbriefen aufgrund eines logisch ausgezeichneten Dokumentes verwendet werden.

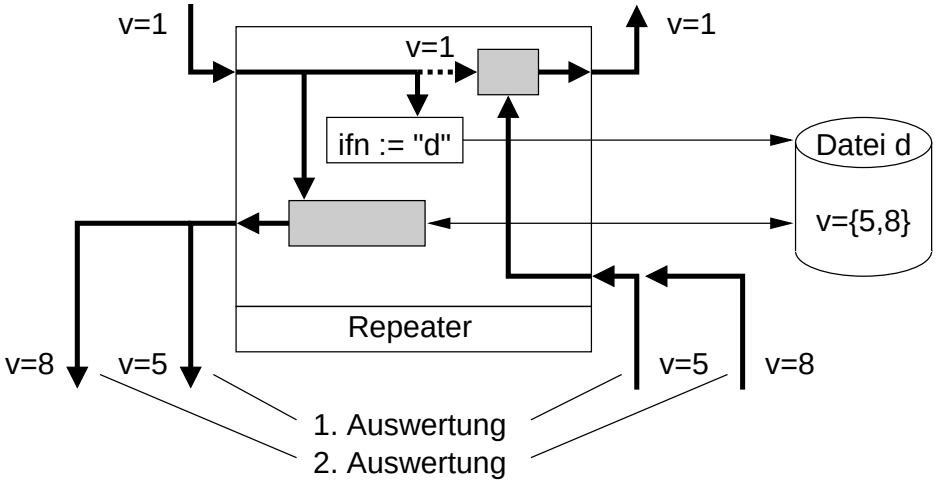

Abbildung 4.19: Spezifikationsvariablenfluss bei einem `Repeater` Auswertungsobjekt: Der Name der Steuerdatei für das `Repeater` Auswertungsobjekt wird in dessen Spezifikationsattribut `inputFileName` (abgekürzt mit `ifn`) gesetzt. Es wird angenommen, dass in der das `Repeater` Auswertungsobjekt steuernden Datei `d` die Spezifikationsvariable `v` eingeführt und gemäß den Vorgaben in der Datei auf die Werte 5 und 8 gesetzt wird. Deshalb wird zunächst eine Auswertung des Unterbaumes mit einem Spezifikationsvariablenfluss gestartet, bei dem `v` auf den Wert 5 gesetzt wird. Im Anschluss daran wird dann ein zweiter Auswertungsdurchlauf mit dem Wert 8 für `v` ausgeführt.

Zuletzt werden nun die **String** Auswertungsobjekte eingeführt. Sie besitzen eine Menge von *internen Eingängen*, an die jeweils beliebige *interne Formatierungen* angeschlossen werden können. Diese können im Rahmen einer wiederholten *lokalen* Auswertung des `String` Auswertungsobjektes durch sich selbst anstelle des `String` Auswertungsobjektes in eine Designstruktur einkopiert werden. In die erweiterte Designstruktur wird das `String` Auswertungsobjekt mit einkopiert, seine einkopierten internen Formatierungen sind dabei als seine Kinder angeordnet. Die Kind-Layoutobjekte eines `String` Auswertungsobjektes werden hinter seinem letzten internen Kind in die Designstruktur einkopiert.

`String` Auswertungsobjekte werten dazu sequentiell für jedes einzelne Zeichen, das in ihrem `input` Spezifikationsattribut enthalten ist, anhand von für die internen Formatierungen gegebenen Bedingungen aus, welche dieser internen Formatierungen in die Designstruktur einkopiert wird. Die Auswertung eines `String` Auswertungsobjektes erfolgt also analog zu einer mehrfachen Auswertung (gesteuert durch eine `while`-Schleife über alle Zeichen der `input` Spezifikationsvariable) einer in üblichen Programmiersprachen verfügbaren `if`-Kaskade.

Für jedes einzelne Zeichen, das in der `input` Spezifikationsvariable des `String` Auswertungsobjektes gesetzt ist, wird eine Auswertung des `String` Auswertungsobjektes durchgeführt. Bei jeder dieser Auswertungen wird im Spezifikationsvariablenfluss vom `String` Auswertungsobjekt eine Menge von Spezifikationsvariablen gesetzt, die natürlich auch in den Bedingungen für die Anwendung der einzelnen internen Formatierungen abgefragt werden können. Als Bedingung könnte z.B. formuliert werden, dass das zu verarbeitende Zeichen ein

bestimmter Buchstabe ist, oder an einer bestimmten Position im `input` Spezifikationsattribut steht. Hierzu führt ein `String` Auswertungsobjekt die Spezifikationsvariablen `character` von der Sorte `String` (enthält das aktuelle logische Zeichen), `char_pos` von der Sorte `Number` (enthält die Position das aktuell verarbeiteten Zeichens innerhalb des `input` Spezifikationsattributes) und `char_pos_b` ebenfalls von der Sorte `Number` (enthält die Position das aktuell verarbeiteten Zeichens innerhalb des `input` Spezifikationsattributes, vom Ende aus berechnet) ein.

Exakt läuft der Auswertungsalgorithmus in einem `String` Auswertungsobjekt folgendermaßen ab (falls überhaupt ein Zeichen im `input` Spezifikationsattribut vorhanden ist):

1. Initialisierung:
   Belege die Spezifikationsvariable `character` mit dem ersten Zeichen aus `input`, die Spezifikationsvariable `char_pos` mit 1 und die Spezifikationsvariable `char_pos_b` mit der Anzahl der Zeichen in `input`.

2. Auswertung `if`-Kaskade:
   Wähle die erste interne Formatierung aus, deren Bedingung sich zu `true` berechnen läßt. Dies ist spätestens bei der letzten internen Formatierung (*Default-Formatierung*) des `String` Auswertungsobjektes der Fall, die immer vorhanden ist (evtl. als *leere* Formatierung).

3. Auswertung:
   Werte die gewählte interne Formatierung aus und kopiere sie in die Designstruktur ein.

4. Fortschaltung:
   Falls `char_pos` kleiner als die Anzahl der Zeichen in `input` ist:
   Besetze `character` mit dem nächsten Zeichen aus `input`, erhöhe `char_pos` um eins, erniedrige `char_pos_b` um eins und fahre bei 2. fort.

Graphisch wird die Verfahrensweise des `String` Auswertungsobjektes in Bezug auf den Spezifikationsvariablenfluss in Abbildung 4.20 dargestellt.

4.4.6 Der logische Fluss in erweiterten Designstrukturen

In Abschnitt 4.4.5 wurde gezeigt, wie mit Hilfe der Spezifikationsvariablen und des Spezifikationsvariablenflusses gezielt die Spezifikationsattribute von Spezifikationsobjekten gesetzt werden können. Diese Möglichkeit der Wertsetzung für Spezifikationsattribute alleine reicht jedoch für eine Designspezifikationsmethodik nicht aus – die eigentlichen Daten eines Dokumentes, die ja in den logischen Attributen der logischen Objekte gespeichert sind, müssen ebenfalls zur Setzung der Spezifikationsattribute herangezogen werden können.

Das typische Beispiel hierfür ist der Text eines Dokumentes, der in [em:] mit Hilfe der logischen Objekte `text` und ihrem logischen Attribut `@input` modelliert wird; er muss zur

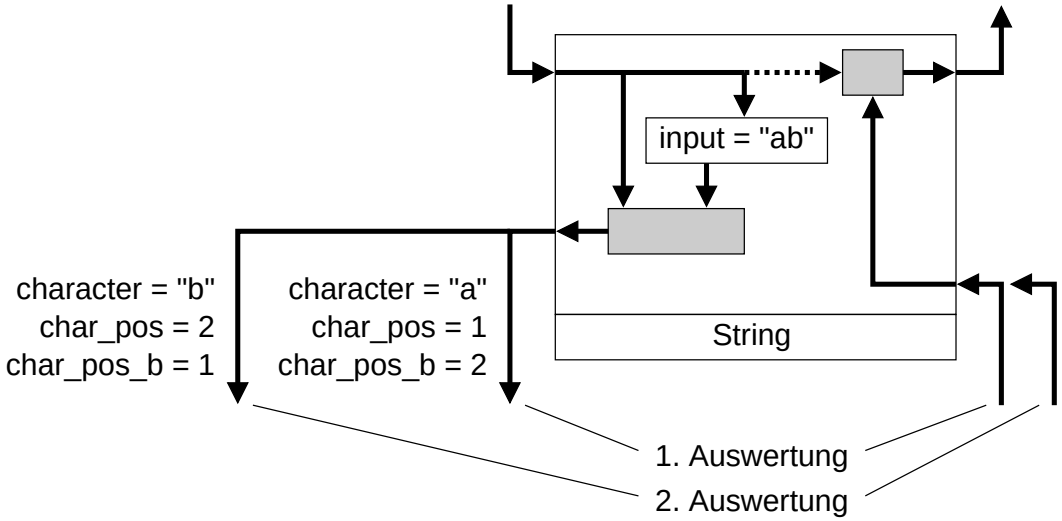

Abbildung 4.20: Spezifikationsvariablenfluss bei einem `String` Auswertungsobjekt: Aufgrund des in seinem Spezifikationsattribut `input` gesetzten Strings `"ab"` manipuliert das `String` Auswertungsobjekt den Spezifikationsvariablenfluss und führt zwei Auswertungsläufe durch, wobei es die Spezifikationsvariablen `character`, `char_pos` und `char_pos_b` manipuliert.

Setzung des Layoutattributes `input` der `Glyphs` Layoutobjekte verwendet werden, um den im abstrakten Dokument gegebenen Text auch im konkreten Dokument zu erhalten.

Das in der Designspezifikationsmethodik [em:] zur Erreichung dieses Zieles (der Verwendbarkeit der Werte von logischen Attributen während einer Designapplikation) gewählte Modell basiert auf der erweiterten Designstruktur, den in ihr vorkommenden logischen Objekten, ihren logischen Attributen, sowie einer sinngemäß übertragenen Auffassung des logischen Flusses der logischen Bäume auf die erweiterte Designstruktur.

Demgemäß verfügt also eine erweiterte Designstruktur ebenso wie ein in [em:] modelliertes logisch ausgezeichnetes Dokument (in Form eines logischen Baumes) über einen logischen Fluss, der die Werte der logischen Attribute in den Unterbäumen der sie einführenden logischen Objekte verfügbar macht. Auf diese Werte kann in den in der erweiterten Designstruktur vorkommenden Spezifikationsobjekten zugegriffen werden. Die Funktionalität eines logischen Objektes in Bezug auf den logischen Fluss wird in Abbildung 4.21 verdeutlicht.

Von der Funktionalität her betrachtet sind die in einer erweiterten Designstruktur vorkommenden logischen Objekte gleichwertig zu den `Inheritance` Auswertungsobjekten. Es ist an dieser Stelle aber wichtig zu betonen, dass die Werte der logischen Attribute im logischen Fluss zur Verfügung gestellt werden, der ein paralleles Konzept zum Spezifikationsvariablenfluss und damit vollkommen unabhängig von diesem ist. Aus diesem Grund können die Werte von logischen Attributen nicht mit Hilfe der Auswertungsobjekte, die nur im Spezifikationsvariablenfluss eine Wirkung verursachen können, im Laufe einer Designapplikation verändert werden. Eine Änderung des Wertes eines logischen Attributes in einer erweiterten Designstruktur ist nur über die vernestete Deklaration eines logischen Attributes mit Hilfe von zwei

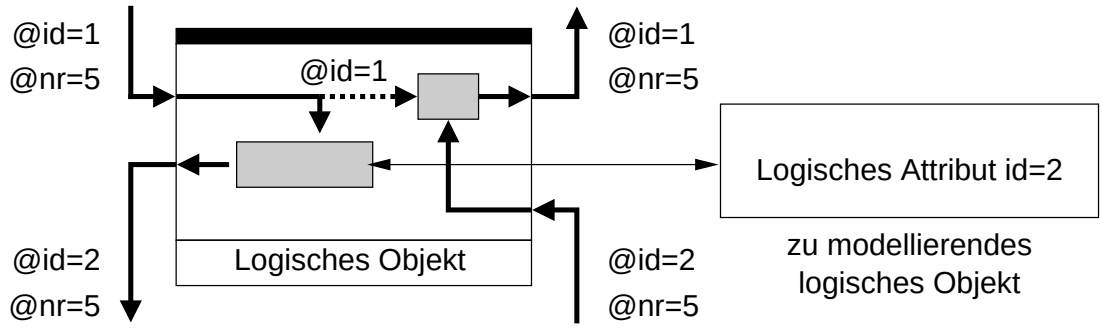

Abbildung 4.21: Logischer Fluss bei einem logischen Objekt in einer erweiterten Designstruktur: Zur besseren Unterscheidbarkeit von Spezifikationsobjekten werden die logischen Objekte von nun an mit einem schwarzen Balken an ihrer Oberseite versehen dargestellt. Der das logische Objekt erreichende logische Fluss führt bereits die logischen Attribute `@nr` und `@id` mit den Werten `5` und `1` mit sich. Das zu modellierende logische Objekt führt erneut das logische Attribut `@id` (mit dem Wert 2) ein. Aus diesem Grund deklariert das in einer erweiterten Designstruktur enthaltene logische Objekt das logische Attribut `@id` und setzt es auf den Wert 2. Dabei verschattet es das gleichnamige Attribut, das bereits weiter oben in der erweiterten Designstruktur eingeführt wurde. Andere nicht verschattete logische Attribute fließen unverändert durch das logische Objekt hindurch. Nach dem Umfließen des logischen Objektes, wenn der logische Fluss das logische Objekt wieder verlässt, sind die Werte der logischen Attribute wieder auf die Werte beim Betreten zurückgesetzt. Die geschilderte Funktionalität des logischen Objektes wird, wie schon bei den Auswertungsobjekten, mit Hilfe der grau hinterlegten Box symbolisiert.

logischen Objekten möglich, die dasselbe logische Attribut besitzen.

Findet während einer Designapplikation ein Zugriff auf ein nicht-deklariertes logisches Attribut statt, so wird in [em:] der leere String als Ergebnis für diesen an sich *illegalen* Zugriff geliefert. Dieses fehlertolerante Verhalten wird in [em:] aufgrund der Tatsache gewählt, dass Designregeln im Verlauf einer Designappplikation unvorhergesehen an Stellen in einem logisch ausgezeichneten Dokument verwendet werden können, die zum Zeitpunkt der Regelaufstellung nicht offensichtlich waren – und somit auch ausserhalb des Deklarationsbereiches von in der Designregel verwendeten logischen Attributen stattfinden können.

In Abbildung 4.22 ist zur Verdeutlichung des logischen Flusses in erweiterten Designstrukturen ein Beispiel einer erweiterten Designstruktur dargestellt, in der fünf logische Objekte zwei verschiedene logische Attribute einführen, wobei es zu einer Verschattung eines dieser logischen Attribute kommt.

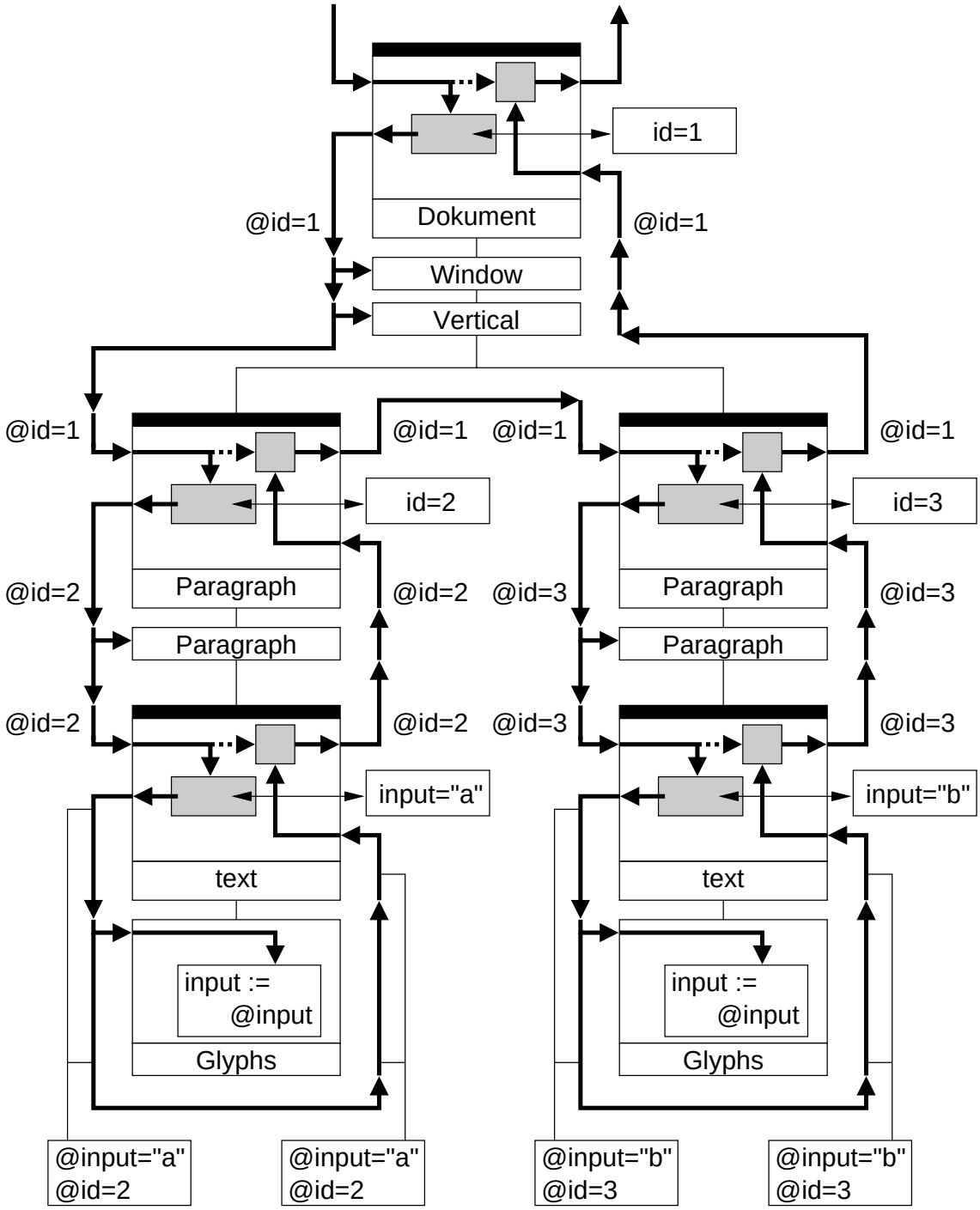

Abbildung 4.22: Logischer Fluss in einer erweiterten Designstruktur: Die logischen Objekte `Dokument` und `Paragraph` deklarieren jeweils das logische Attribut `id`, wodurch es beim logischen Fluss an beiden logischen `Paragraph` Objekten zu einer Verschattung dieses logischen Attributes kommt. Das logische `text` Objekt führt das logische Attribut `@input` ein, das in den `Glyphs` Layoutobjekten Verwendung finden kann.

4.4.7 Zusammenfassung

In Abschnitt 4.4 wurde vorgestellt, wie man in der Designspezifikationsmethodik [em:] aufgrund eines gegebenen abstrakten Dokumentes zu einem dieser Vorgabe entsprechenden konkreten Dokument gelangt. Die prinzipielle Vorgehensweise, die hierzu in [em:] gewählt wird, ist zunächst die Abstützung auf zu Formatierungen zusammengefaßte Layoutobjekte, um das Layout von einzelnen logischen Objekten isoliert beschreiben zu können. Diese Formatierungen werden, mit einem Umgebungsprädikat zur Ermöglichung einer kontextabhängigen Anwendung versehen, zur Bildung von Designregeln verwendet. Deren Vereinigung zu einer Menge wiederum ergibt eine Designspezifikation, die im Rahmen einer Designapplikation auf ein logisch ausgezeichnetes Dokument angewendet werden kann.

Es hat sich bei der Entwicklung der [em:]-Methodik gezeigt, dass erst die Ergänzung der Designregeln um Konnektoren den sinnvollen Zusammenbau von einzelnen isolierten Formatierungen zu einer Designstruktur, die statisch einen Formatierpozess beschreibt, ermöglicht. Dieser Zusammenbau wird durch das Zusammenwirken des verwendeten Matching-Algorithmus zur Auswahl der anzuwendenden Designregel und der in den Designregeln vorkommenden Konnektoren zur weiteren Auswahl, welche logischen Objekte verarbeitet werden sollen, zu einem verteilt gesteuerten Prozess.

Die mit Hilfe der bisher in dieser Zusammenfassung beschriebenen Hilfsmittel konstruierbaren Designspezifikationen beschreiben zwar das prinzipielle Layout von Dokumenten, zur Erzielung qualitativ hochwertiger Resultate reichen sie jedoch nicht aus. Hierzu müssen auch noch die Layoutattribute gesetzt werden können, die das Verhalten der in einer Designstruktur vorkommenden Layoutobjekte steuern. Dies wird in [em:] durch die Einführung von Spezifikationsvariablen, den sie manipulierenden Auswertungsobjekten und dem dabei entstehenden Spezifikationsvariablenfluss realisiert. Mit Hilfe dieses Spezifikationsvariablenflusses ist aber nicht nur die ausschließliche Setzung von Layoutattributen möglich. Vielmehr können mit Hilfe dieses Gestaltungsinstruments noch weitere Eigenschaften im Rahmen einer Designapplikation gesteuert werden. Dies sind z.B. benutzerdefinierte Nummerierungen sowie abgeleitete Dokumentbestandteile. Zur Verdeutlichung der Möglichkeiten hierbei wurde als Hilfskonstrukt in [em:] die erweiterte Designstruktur eingeführt, die die mit Hilfe des Spezifikationsvariablenflusses in einer Designstruktur ermöglichte Propagierung von Layouteigenschaften den Nutzern leicht verständlich zugänglich machen soll.

Der letzte noch offene Punkt, der Zugriff auf die Dokumentdaten im Rahmen einer Designapplikation, wird in [em:] mit Hilfe des logischen Flusses, der ebenfalls wie der Spezifikationsvariablenfluss in der erweiterten Designstruktur vorhanden ist, erklärt. Erst gemeinsam reichen der logische Fluss und der Spezifikationsvariablenfluss zur Setzung von Werten für die Attribute von Layoutobjekten aus, um qualitativ hochwertige Designspezifikationen zu erhalten.

4.5 Das Multi-View-Konzept

Die bisherigen Möglichkeiten, die die Designspezifikationsmethodik [em:] ihren Nutzern zur Erstellung einer Designspezifikation an die Hand gibt, ermöglichen bei einer *geradlinigen* Ver-

wendung ausschließlich eine *einfache* Überführung eines abstrakten Dokumentes in ein *gleichwertiges* konkretes Dokument. Dies bedeutet, dass im Wesentlichen jeder Bestandteil des abstrakten Dokumentes nur eine Repräsentation im korrespondierenden konkreten Dokument erfährt. Hinzu kommen lediglich wenige Anreicherungsmöglichkeiten, wie z.B. Nummerierungen und generierte Bestandteile, die mit Hilfe einer gezielten Manipulation des Spezifikationsvariablenflusses erreicht werden können.

Eine Designspezifikationsmethodik muss jedoch mehr leisten, als nur in dieser einfachen Weise das Layout eines Dokumentes zu spezifizieren. Sie muss auch Teile des redaktionellen Prozesses, wie er in 3.2 untersucht wurde, übernehmen können. Ein typisches Beispiel hierfür ist die Erstellung der Titelei eines Dokumentes und somit die Möglichkeit, aufgrund eines abstrakten Dokumentes, in dem kein explizites Inhaltsverzeichnis enthalten ist, ein konkretes Dokument zu erstellen, in dem sowohl ein Inhaltsverzeichnis als auch der eigentliche Textteil enthalten sind.

Dies ist jedoch bisher in [em:] nur schwer (und insbesondere nur unter *Missbrauch* von Konstrukten der [em:]-Methodik) möglich. Die logischen Objekte eines gegebenen logisch ausgezeichneten Dokumentes werden im Wesentlichen im Verlauf einer Designapplikation nur einmal in Form eines depth-first Durchlaufes verarbeitet, was für einen praxisgerechten Einsatz der Designspezifikationsmethodik [em:] eine große Einschränkung darstellt. Der Applikationspfad entspricht in etwa der linearisierten Struktur des abstrakten Dokumentes.

4.5.1 Sichten auf ein Dokument

Zur Abhilfe dieses Mankos wird in [em:] nun zunächst der intuitiv leicht verständliche Begriff eines *Views auf ein Dokument* (einer Sicht auf ein Dokument) eingeführt, der sowohl auf abstrakten als auch auf konkreten Dokumenten gültig ist.

Definition 54 (View auf ein Dokument) *Ein View auf ein Dokument ist die Betrachtung eines Dokumentes unter bestimmten vorgegebenen Aspekten. Dazu können gemäß den gewählten Aspekten gewisse Bestandteile des Dokumentes ausgeblendet, andere generierte Bestandteile jedoch mit hinzugenommen werden. Ein View auf ein Dokument erzeugt ein neues Dokument.*

Im Sinne dieser Definition sind bisherige Designspezifikationen, die in der Designspezifikationsmethodik [em:] erstellt werden können, als *Single-View* Designspezifikationen zu bezeichnen – sie können nur konkrete Dokumente erzeugen, die einen einzigen View darstellen.

Die Idee der Views und ihre Realisierung in [em:] wird im Folgenden mit Hilfe der Pipeline-Metapher verdeutlicht. Die Grundidee hierzu ist einfach: Bei einer bisherigen Designapplikation wurde logischen Objekten mit Hilfe von Designregeln Formatierungen zugeordnet, mit deren Hilfe ein Layout-Pipeline-System aufgebaut wurde. In diesem Layout-Pipeline-System sind dann alle Layoutbestandteile mit Hilfe eines einzigen Layoutflusses transportiert worden. In zukünftigen Designspezifikationen hingegen, die *erweiterte Designregeln* enthalten, wird es möglich sein, mehrere unabhängige Layout-Pipeline-Systeme aufzubauen, die zur besseren Veranschaulichung als mit verschiedenfarbigen Rohren aufgebaut betrachtet werden. In diesen

Layout-Pipeline-Systemen fließen weitere Layoutflüsse, die untereinander und ebenfalls von dem ursprünglichen einzigen Layoutfluss, unabhängig sind. Auf diese Weise lassen sich beliebig viele Layoutflüsse erzeugen und somit logische Objekte beliebig oft im Rahmen einer Designapplikation im Sinne einer Aufgabenstellung (und somit eines Views) verwenden. Jedes dieser Layout-Pipeline-Systeme realisiert einen Single-View auf ein Dokument.

Mit diesem Vorgehen erreicht man nun zwar die *Mehrfachverwendung* von logischen Dokumentbestandteilen und kann insgesamt mehrere Views auf ein Dokument realisieren. Dies alleine steigert aber noch nicht die Mächtigkeit der Designspezifikationsmethodik [em:]. Für den Fall, dass alle Layout-Pipeline-Systeme voneinander unabhängig sind, bleiben auch die Layoutflüsse immer voneinander getrennt; die Layout-Pipeline-Systeme erzeugen vollkommen voneinander unabhängige Ausgaben. Es wäre also ebenso gut möglich, nacheinander mehrere Designapplikationen auszuführen, die jeweils nur ein Layout-Pipeline-System aufbauen. Die Views können nicht miteinander kombiniert werden.

In einer Publikation (z.B. einem Buch) sind aber in der Regel mehrere Sichten (Inhaltsverzeichnis, Textteil) auf ein- und dasselbe abstrakte Dokument zu einer Einheit zusammengefaßt. Es fehlt somit noch eine Möglichkeit, mit deren Hilfe in [em:] die bisher getrennten Layout-Pipeline-Systeme an geeigneten Stellen zusammengefaßt werden können, um eine neue Einheit (ein neues Layout-Pipeline-System) zu bilden, das einen *Multi-View* auf ein Dokument darstellt.

4.5.2 Technische Umsetzung

Realisiert wird diese fehlende Möglichkeit zur Realisierung eines Multi-Views in [em:] in zwei Schritten. Zunächst wird es technisch ermöglicht, innerhalb einer Designapplikation mehrere unabhängige Designstrukturen aufzubauen, die mehrere voneinander unabhängige Views darstellen. Dies geschieht über eine Erweiterung der Definition der Designregeln:

Definition 55 (Erweiterte Designregel) *Eine erweiterte Designregel ist eine Designregel gemäß Definition 43, die um eine Kennung, für welchen View sie gültig ist, erweitert wird.*

Aufgrund dieser Erweiterung der Designregel-Definition wird es nötig, auch den Formalismus zur Aufschreibung einer Designregel zu ändern. Unter der Prämisse, dass die Kennungen für Views natürliche Zahlen sind, kann eine Designregel, die einem logischen Objekt `obj` bei Erfüllung des Umgebungsprädikates `p` für den View `1` eine Formatierung `f`, die nur aus einem Layoutobjekt `lo` besteht, zuordnet, wie folgt formuliert werden:

$$\begin{array}{lll} \text{obj} & \leftarrow_p^1 & (\text{lo} \\ & &)\text{lo} \end{array}$$

In diesem Kapitel bereits erfolgte Erklärungen, die sich auf den Begriff Designregel beziehen, können allgemein einfach sinngemäß auf die Definition der erweiterten Designregeln angepasst werden. Die zentrale Stelle, an der eine Anpassung erfolgen muss, ist die Designapplikation: Eine Designapplikation mit erweiterten Designregeln wird nun im Folgenden immer

für einen bestimmten, von den Nutzern vorgebbaren, View `n` gestartet. Dieser View `n` gibt vor, welche Designregeln überhaupt beim Matching-Algorithmus betrachtet werden müssen (nämlich nur die, die für den gerade gültigen View `n` aufgestellt sind). Das restliche Vorgehen bei einer Designapplikation bleibt unverändert: Wie bereits beschrieben, stoßen dann für den weiteren Aufbau der Designstruktur die in einer gewählten Designregel vorkommenden Konnektoren weitere Aufrufe von Matching-Algorithmen an.

Dies ist die zweite entscheidende Stelle, die in [em:] geändert wird, damit die bisher unabhängigen Views (in Form von Designstrukturen) zusammengebracht werden können. Konnektoren können den View, unter dem ein Dokumentteil aktuell verarbeitet wird, ändern.

Definition 56 (Erweiterter Konnektor) *Ein erweiterter Konnektor ist ein Konnektor gemäß Definition 47. Zusätzlich zu den dort getroffenen Entscheidungen kann ein erweiterter Konnektor den View festlegen, der für die Verarbeitung seines Unterbaumes gelten soll.*

Somit kann man zusammenfassen, dass Designregeln gemäß den zwei neu eingebrachten Änderungen in [em:] (erweiterte Designregeln und erweiterte Konnektoren) zur Realisierung der Multi-View-Fähigkeit von [em:] zunächst für einen bestimmten View aufgestellt werden, dessen Design sie realisieren sollen. Dieser View gibt den inhaltlichen Aspekt vor, für welche Aufgabe ein Dokumentteil aufbereitet werden soll und bestimmt deshalb das Aussehen der Formatierung. Weiterhin kann von nun an in Designregeln für jeden einzelnen der in ihnen vorkommenden Konnektoren festgelegt werden, gemäß welchem View das logisch ausgezeichnete Dokument für ihn weiter verarbeitet werden soll.

Mit Hilfe dieser Vorgehensweise können Designregeln für Views, die bereits an einer Stelle definiert wurden, an einer beliebigen anderen Stelle in einer Designspezifikation erneut ohne einen weiteren Spezifikationsaufwand durch einen einfachen View-Wechsel mit Hilfe eines Konnektors eingesetzt werden.

Diese soeben angesprochene Eigenschaft der Designspezifikationsmethodik [em:] wird gemäß folgender Designspezifikation in Abbildung 4.23 anhand eines logisch ausgezeichneten Dokumentes erklärt, das kein explizites Inhaltsverzeichnis enthält.

```
Dokument <-3 (Window
             (Vertical
             (Konnektor
             View:=1
             )Konnektor
             (Konnektor
             View:=2
             )Konnektor
             )Vertical
             )Window

Kapitel  <-1 (Konnektor
             View:=1
             )Konnektor

Kapitel  <-2 (Konnektor
             View:=2
             )Konnektor
```

In obigem Ausschnitt aus einer Designspezifikation sind drei Designregeln für drei verschiedene Views angegeben. Mit Hilfe eines Views `1` wird für ein Dokument das Design für dessen Inhaltsverzeichnis erstellt, mit Hilfe von View `2` das Design für seinen Textteil. In View `3` können die beiden Views mit Hilfe einer Formatierung, die zwei Konnektoren enthält, die auf den View `1` bzw. `2` umschalten, wiederverwendet werden.

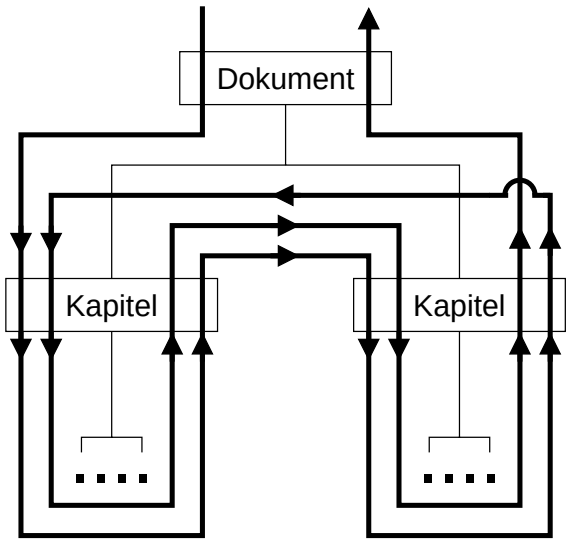

Abbildung 4.23: Applikationspfad bei mehrfacher Verwendung von Dokumentbestandteilen: Da der Inhalt des Dokumentes einmal zur Erzeugung eines Inhaltsverzeichnisses und einmal zur Erzeugung des Textteiles verwendet wird, durchläuft der Applikationspfad das Dokument von Kapitelebene an zweimal.

4.6 Das Beamer-Konzept

4.6.1 Layoutbestandteile in Querverweisen

Mit den bisher in der Designspezifikationsmethodik [em:] angebotenen Konzepten ist es nun möglich, ein gegebenes Dokument mehrfach komplett von vorne bis hinten (oder auch in kleineren Bereichen) zu verarbeiten, um darauf basierend eine Designstruktur zu erstellen. Es können so z.B. konkrete Dokumente erzeugt werden, die vor dem eigentlichen Textteil ein Inhalts- und ein Abbildungsverzeichnis enthalten. Weiterhin ist es auch möglich, die in einer derartigen Designstruktur vorkommenden Abbildungen mit Hilfe von durch die Benutzer definierbaren Zählern durchzunummerieren, und dies konsistent sowohl im Abbildungsverzeichnis als auch im Textteil.

Es ist jedoch in [em:] bisher nicht möglich, diese Nummerierung im Rahmen von Querverweisen zur Bezugnahme zur verwenden. Dies ist jedoch eine typische Aufgabenstellung in herkömmlichen Publikationen. So findet man in vielen wissenschaftlichen Artikeln als Realisierung eines Verweises den Text „siehe Abbildung ...“ als Hinweis für die Leser, welche

Abbildung an einer gewissen Textstelle referenziert wird. Ein Dokumentausschnitt, der eine derartige Situation darstellt, ist in Abbildung 4.24 dargestellt.

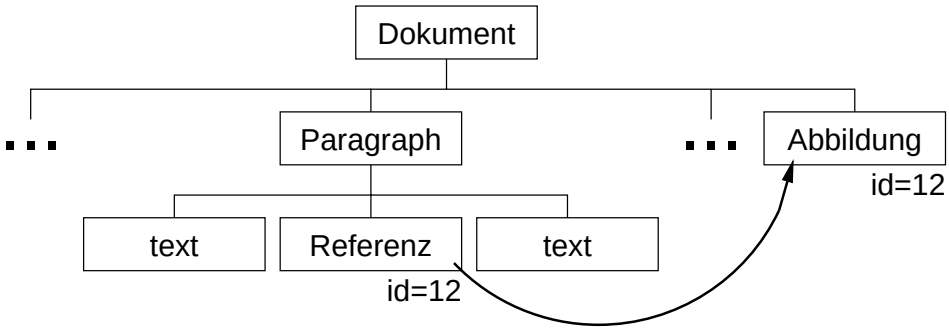

Abbildung 4.24: Beispieldokument mit einem Querverweis: In dem abgebildeten Dokument wird von einem Paragraphen aus eine Referenz auf eine Abbildung mit der `id 12` gesetzt.

Präziser formuliert ist somit das in [em:] noch nicht gelöste Problem der Transport von Layoutbestandteilen weg von einer referenzierten Stelle hin zu der bezugnehmenden Stelle im Rahmen einer Designapplikation. Anzumerken ist dabei, dass dieser Transport von Layoutbestandteilen bei mehreren Referenzen auf ein und dieselbe Stelle auch mehrfach nötig sein kann.

Das Beamen

Gelöst wird diese Aufgabenstellung in [em:] mit Hilfe eines Konzeptes, das den meisten Lesern vom Prinzip her wohl aus Science Fiction Filmen bekannt ist: Dem *Beamen* von Layoutbestandteilen von einer Stelle an eine andere, wobei unter Umständen eine Vervielfachung der gebeamten Layoutbestandteile eintreten kann. Wichtig zu beachten ist dabei jedoch, dass Layoutbestandteile, bevor sie gebeamt werden können, auch erzeugt worden sein müssen.

Zur Realisierung dieses Beamer-Konzeptes werden in [em:] zwei neue Spezifikationsobjekte zur Verfügung gestellt, ein `Beamer` und ein `Materializer` Spezifikationsobjekt. Das `Beamer` Spezifikationsobjekt fasst die bei seiner Auswertung unter ihm entstehende Struktur von Spezifikationsobjekten zu einer Einheit zusammen und fügt sie in einen benannten *Beamer-Fluss* ein, der mit Hilfe des `key` Spezifikationsattributes beim `Beamer` Spezifikationsobjekt angegeben werden kann. Als Gegenstück zu einem `Beamer` Spezifikationsobjekt kann das `Materializer` Spezifikationsobjekt einen benannten Beamer-Fluss *anzapfen* (ebenfalls über das `key` Spezifikationsattribut festlegbar) und die darin vorhandenen Spezifikationsobjektstrukturen verwenden – sie werden anstelle des `Materializer` Spezifikationsobjektes in die Designstruktur einkopiert.

Im Gegensatz zur üblichen Vorstellung des *Beamens* werden die von einem `Beamer` Spezifikationsobjekt aufgesammelten Spezifikationsobjekte aber nicht ausschließlich über den `Beamer` verschickt, sondern tragen zusätzlich wie bisher zum Aufbau der berechneten Designstruktur bei, so als ob das `Beamer` Spezifikationsobjekt über ihnen nicht präsent wäre.

Es ist möglich, dass mehrere voneinander unabhängige `Beamer` Spezifikationsobjekte Inhalte in ein und denselben Beamer-Fluss einfügen. In diesem Fall fügt ein `Materializer` Spezifikationsobjekt, das diesen Beamer-Fluss verwendet, alle im Beamer-Fluss enthaltenen Inhalte in der Reihenfolge ein, wie sie durch `Beamer` Spezifikationsobjekte in den Beamer-Fluss eingefügt wurden. Die zum mehrfachen Einfügen in einen Beamer-Fluss umgekehrte Situation ist ebenfalls möglich – ein mehrfaches Entnehmen aus einem Beamer-Fluss. Dies ist dann der Fall, wenn mehrere `Materializer` Spezifikationsobjekte ein- und denselben Beamer-Fluss anzapfen. In diesem Fall stellt sich für jedes `Materializer` Spezifikationsobjekt die Situation derart dar, als ob es das einzige Spezifikationsobjekt ist, das den Beamer-Fluss anzapft. Der Beamer-Fluss wird also vervielfacht.

Auswertung der gebeamten Spezifikationsobjekte

Bei der Verwendung von `Beamer` Spezifikationsobjekten stellt sich weiterhin die Frage, mit welchen Werten die Spezifikationsvariablen und Spezifikationsattribute der durch sie aufgesammelten Spezifikationsobjekte besetzt werden sollen. Hierbei sind prinzipiell zwei gegensätzliche Möglichkeiten offen. Zum einen können die Werte zu dem Zeitpunkt gesetzt werden, zu dem ein `Beamer` Spezifikationsobjekt seine Kind-Spezifikationsobjekte auswertet und einsammelt; die Spezifikationsobjekte werden in diesem Fall sozusagen *fertig ausgebildet* transportiert. Zum anderen ist es aber auch möglich, die Werte erst dann zu berechnen, wenn die aufgesammelten Spezifikationsobjekte mit Hilfe eines `Materializer` Spezifikationsobjektes in eine Designstruktur einkopiert werden; in diesem Fall werden die Spezifikationsobjekte noch an die Zielumgebung *anpassungsfähig* transportiert.

In [em:] wurde ein Mittelweg zwischen diesen beiden Extremen gewählt, der realen Anforderungen gerecht werden kann. Als Beispiel (siehe auch Abbildung 4.24) hierzu dient ein `Glyphs` Layoutobjekt, das die Nummer einer Abbildung von einem logischen Objekt `Abbildung` zu einem logischen Objekt `Referenz` mit Hilfe eines `Beamer` Spezifikationsobjektes transportieren soll. In diesem Fall muss der Wert der `input` Layoutvariable des `Glyphs` Layoutobjektes beim Einsammeln durch das `Beamer` Spezifikationsobjekt gesetzt werden (z.B. auf die Nummer der Abbildung), der Wert der `fontSize` Layoutvariable sollte jedoch erst beim Einkopieren in die Designstruktur berechnet werden, um den selben Wert wie der umgebende Text zu erhalten.

Der erste Fall der sofortigen Wertsetzung beim Einsammeln wird in [em:] mit dem Begriff *immediate*-Auswertung bezeichnet, der zweite Fall einer verzögerten Auswertung mit *deferred*-Auswertung. Bei der Angabe der Berechnungsvorschrift für Spezifikationsattribute in [em:] kann im Falle einer Auswertung unterhalb eines `Beamer` Spezifikationsobjektes zusätzlich zu den bisher vorgestellten Möglichkeiten angegeben werden, ob die Wertsetzung gemäß dem *immediate*- oder dem *deferred*-Schema erfolgen soll.

Beispiel

Eine Designspezifikation zu obigem Beispiel des Transports einer Abbildungsnummer von einem logischen Objekt `Abbildung` zu einem logischen Objekt `Referenz` (siehe Abbil-

dung 4.24) kann in [em:] schematisch wie in folgendem Auszug aus einer Designspezifikation realisiert werden. Dabei ist es wichtig, dass in irgendeiner Form View `1` vor View `2` berechnet wird, damit im Beamer-Fluss mit Kennung `12` (berechnet aufgrund des Wertes des logischen Attributes `id`) der Inhalt, der bei dem logischen Objekt `Abbildung` erzeugt wird, schon vorhanden ist, wenn er bei dem logischen Objekt `Referenz` verwendet werden soll.

```
Abbildung ←1 (Beamer
             key:=@id
             (Glyphs
             (immediate) input:="..."
             (deferred) fontSize
             )Glyphs
             )Beamer
```

```
Referenz ←2 (Materializer
            key:=@id
            )Materializer
```

Versucht man das Beamer-Konzept in das Konzept der Layout-Pipeline-Systeme zu integrieren, so ergeben sich Rohre, die quer vor dem Rest eines Layout-Pipeline-Systems vorbeilaufen – sie realisieren eben Querverweise. Mit Hilfe des Beamer-Konzeptes lassen sich jedoch nicht nur Querverweise realisieren, sondern auch Bezüge zu beliebigen Verzeichnissen. Eine Gemeinsamkeit hierbei, die bei allen aufgeführten Beispielen gilt, ist die Tatsache, dass über einen Beamer-Fluss in der Regel immer nur ein Inhaltselement fließt. Dies ändert sich jedoch, wenn das Beamer-Konzept zur Realisierung von Indexen verwendet wird.

4.6.2 Layoutbestandteile in Indexen

Bei der Verwendung des Beamer-Konzeptes zur Realisierung von Indexen fließt über einen Beamer-Fluss nicht nur ein Inhaltselement, sondern genau so viele, wie Einträge in einem Index erstellt werden sollen. Folgendes Beispiel soll dies verdeutlichen (siehe hierzu auch Abbildung 4.25): Jeder Indexeintrag erzeugt mit Hilfe eines `Beamer` Spezifikationsobjektes aufgrund der Layoutobjektstruktur seiner Kinder ein Inhaltsobjekt für den Beamer-Fluss `1`. Dieser Beamer-Fluss `1` wird mit Hilfe eines `Materializer` Spezifikationsobjektes am logischen `Index` Objekt aufgenommen und kann zur Erstellung eines Indexes verwendet werden. Hierbei stellt sich jedoch die Frage, wie der Inhalt des ankommenden Beamer-Flusses sortiert werden kann, um diese übliche Eigenschaft eines Indexes zu erreichen.

Die Lösung zu dieser Aufgabenstellung liegt in [em:] in der Einführung eines neuen Spezifikationsobjektes `SortingKey`. Dieses neue Spezifikationsobjekt arbeitet nur mit `Materializer` Spezifikationsobjekten zusammen und hat die Aufgabe, für ein Inhaltsobjekt eines Beamer-Flusses festzulegen, wie dieses Inhaltsobjekt im Vergleich zu anderen Objekten einsortiert werden soll – es stattet die Inhaltsobjekte mit Sortierkriterien aus. Dazu verfügt ein `SortingKey` Spezifikationsobjekt über ein Spezifikationsattribut `sortingKey`, über das der gewünschte Wert für das Sortierkriterium angegeben werden kann.

Das den Beamer-Fluss `1` empfangende `Materializer` Spezifikationsobjekt kann somit aufgrund eines in einem Inhaltsobjekt vorhandenen `SortingKey` Spezifikationsobjektes eine

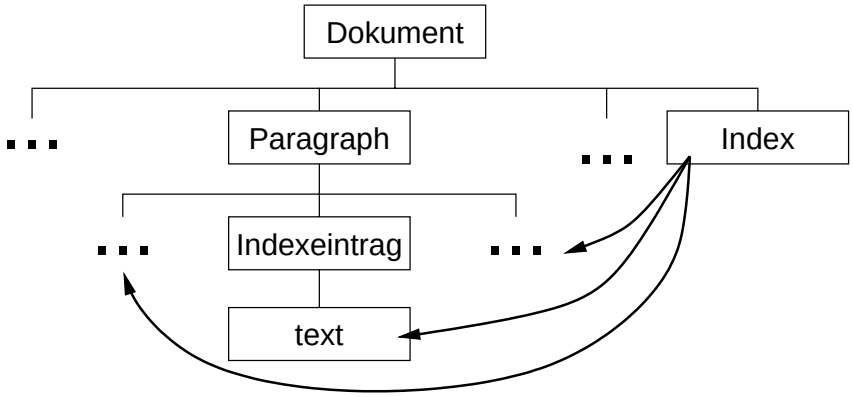

Abbildung 4.25: Beispieldokument zur Erstellung eines Indexes: Vom logischen `Index` Element aus wird implizit Bezug auf alle im Dokument vorkommenden Indexeinträge genommen.

Sortierung des empfangenen Flusses vornehmen. Mit Hilfe des Spezifikationsattributes `sortingOrder` des `Materializer` Spezifikationsobjektes kann dabei sogar angegeben werden, ob die Inhaltsobjekte des empfangenen Flusses auf- oder absteigend sortiert werden sollen.

4.7 Compoundobjekte

In Abschnitt 4.2.2 wurden die vier Kriterien Aufgabenorientiertheit, Einfachheit, Minimalität und Vollständigkeit für Layoutobjekt-Basismengen aufgestellt. Diese vier Kriterien bringen, wie ausführlich erläutert wurde, eine Anzahl von methodischen Vorteilen mit sich. So helfen sie zum einen mit, die Designspezifikationsmethodik [em:] für deren Nutzer einfach zu gestalten. Zum anderen wird durch sie aber auch das in [em:] spezifizierbare Design für Dokumente sehr umfassend.

Es ist jedoch zu beachten, dass insbesondere die beiden Forderungen der Einfachheit und Minimalität an Layoutobjekte zu sehr *elementaren* und *spartanischen* Layoutobjekten führen[36]. Komplexe Designkonstrukte müssen deshalb in [em:] durch eine Kombination von mehreren Basismengen-Layoutobjekten erzielt werden. Dies ist insbesondere dann eine *unangenehme* Eigenschaft von [em:], wenn man an eine eventuell nötige Mehrfachverwendung einer derartigen Kombination von Spezifikationsobjekten in verschiedenen Designregeln denkt, die nur durch ein identisches Kopieren der Struktur durch die Nutzer möglich ist. Nach Einführung der Auswertungsobjekte zur gezielten Setzung der Werte von Spezifikationsattributen wird diese Situation sogar noch umständlicher, da die Anzahl der in einer Designregel enthaltenen Spezifikationsobjekte noch größer wird. Auch ist die Verwendung einer großen Anzahl von Spezifikationsobjekten zur Erzielung eines (wenn auch komplexen) Layouteffektes nicht besonders übersichtlich (wobei sich jedoch die Frage stellt, ob dies bei einem komplexen Effekt nicht legitim ist).

[36]Durch dieses Vorgehen wird es aber jedoch überhaupt erst ermöglicht, auch bei Entwurf der Designspezifikationsmethodik unvorhergesehene Layoutideen zu realisieren.

Es ist somit angezeigt, in [em:] einen Mechanismus anzubieten, der es zulässt, eine Struktur von Spezifikationsobjekten zu einer funktionalen Gruppe mit Hilfe eines neuen Spezifikationsobjektes zusammenzufassen. Dieser Mechanismus wird in [em:] mit Hilfe der sogenannten Compoundobjekte realisiert:

Definition 57 (Compoundobjekt) *Ein Compoundobjekt ist ein Spezifikationsobjekt, das eine beliebige interne Formatierung verwaltet, die mit seiner Hilfe in eine Designstruktur einkopiert werden kann. Für die Steuerung des individualisierten Einkopierens der internen Formatierung in eine Designstruktur kann das Compoundobjekt einen lokalen Spezifikationsvariablenfluss berechnen, der mit Hilfe einer Liste von lokalen Spezifikationsvariablen initialisiert werden kann, über die das Compoundobjekt verfügt.*

4.7.1 Der Spezifikationsvariablenfluss in Compoundobjekten

Wenn die durch ein Compoundobjekt `c` verwaltete interne Formatierung `f` aufgrund einer Verwendung von `c` in einer zu applizierenden Designregel in eine Designstruktur einkopiert werden soll, müssen für die Spezifikationsvariablen und Spezifikationsattribute der in `f` vorkommenden Spezifikationsobjekte Werte berechnet werden. Für diese Berechnung wird, wie allgemein in [em:] üblich, der Spezifikationsvariablenfluss verwendet.

Abstützung auf den globalen Spezifikationsvariablenfluss

Im einfachsten Fall werden zur Berechnung der Werte für die Spezifikationsvariablen und Spezifikationsattribute der Spezifikationsobjekte in `f` der an dieser Stelle global verfügbare Spezifikationsvariablenfluss verwendet. Bei diesem Vorgehen fallen zwei Punkte auf: Zum einen kann keine Individualisierung des Compoundobjektes erfolgen, da durch den Rückgriff auf den globalen Spezifikationsvariablenfluss schlichtweg keine Parameterliste für das Compoundobjekt verwendet wird. Zum anderen kann der globale Spezifikationsvariablenfluss aufgrund von in `f` vorkommenden Auswertungsobjekten manipuliert werden.

Dies führt zu unvorhersehbaren Effekten auf dem Spezifikationsvariablenfluss, die man dem Compoundobjekt `c`, das ja nach außen hin eine monolithische Einheit darstellt, nicht ansieht. Im Gegensatz zu allen anderen Spezifikationsobjekten, deren prinzipielle Wirkung auf den Spezifikationsvariablenfluss klar festgelegt[37] ist (siehe 4.4.5), kann ein Compoundobjekt prinzipiell auf jede Variable im Spezifikationsvariablenfluss demnach in vierfach-verschiedener Weise Auswirkungen haben (Abbildung 4.26 stellt die möglichen Auswirkungen graphisch dar):

1. Es verändert den Spezifikationsvariablenfluss nicht und verhält sich daher wie ein herkömmliches Layoutobjekt.

[37] Ausnahmen hiervon bilden nur die `Inheritance` und `Modifier` Auswertungsobjekte mit leeren Spezifikationsvariablen-Listen sowie das `Modifier` Auswertungsobjekt mit einer Spezifikationsvariable, die lediglich gleich sich selbst gesetzt wird (z.B. `x:=x`), was keine Wirkung auf den Spezifikationsvariablenfluss hat.

2. Es wirkt wie ein `Inheritance` Auswertungsobjekt und verändert deshalb den Wert einer Spezifikationsvariablen im Spezifikationsvariablenfluss beim Fortfahren. Ausserdem blockt es Änderungen der betroffenen Spezifikationsvariable, die im Unterbaum vorgenommen werden, nach oben hin ab (siehe Abbildung 4.17 auf Seite 125). Dies bedeutet, dass in der internen Formatierung des Compoundobjektes mindestens ein `Inheritance` Auswertungsobjekt vorkommt.

3. Es wirkt wie ein `Modifier` Auswertungsobjekt und verändert daher den Wert einer Spezifikationsvariablen im Spezifikationsvariablenfluss erst beim Verlassen. Dies bedeutet, dass in der internen Formatierung mindestens ein `Modifier` Auswertungsobjekt vorkommt.

4. Es wirkt sowohl wie ein `Inheritance` als auch wie ein `Modifier` Auswertungsobjekt. Somit verändert es den Wert einer Spezifikationsvariablen im Spezifikationsvariablenfluss sowohl beim Fortfahren als auch beim Verlassen. Dabei ist nur die Kombination interessant, bei der das `Modifier` Auswertungsobjekt oberhalb des `Inheritance` Auswertungsobjektes angeordnet ist.

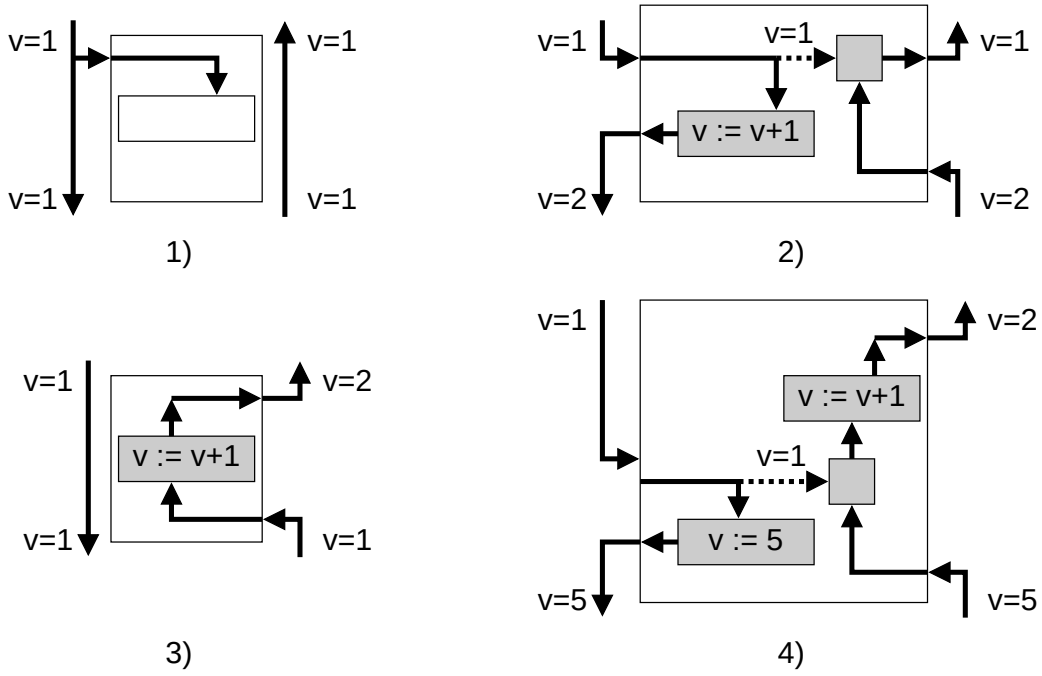

Abbildung 4.26: Die vier Arten möglicher Auswirkungen eines Compoundobjektes auf den Spezifikationsvariablenfluss bezüglich einer Variablen `v`: Die Fälle sind wie in obiger Liste nummeriert.

Es wäre ein funktional einschränkender Eingriff in [em:], wenn man diese Abstützung der Compoundobjekte auf den globalen Spezifikationsvariablenfluss untersagte. Insbesondere wäre es nicht mehr möglich, globale Zähler (z.B. zur Überschriftsnummerierung) von Compoundobjekten aus zu manipulieren. Aus diesem Grund wird diese Option in [em:] ausdrücklich als Gestaltungsmöglichkeit beibehalten. Sie muss jedoch mit Vorsicht eingesetzt werden.

Erzeugung eines lokalen Spezifikationsvariablenflusses

Die Individualisierung der durch ein Compoundobjekt `c` verwalteten internen Formatierung `f` ist ein aufgrund der Entstehungsidee der Compoundobjekte wichtiger Punkt: Compoundobjekte verkörpern bei ihrer Instantiierung als ein neues Spezifikationsobjekt, dessen Funktionalität durch `f` realisiert wird, eine Einheit, die in gewissen Fällen über einen eigenständigen Parametersatz steuerbar sein sollte. Dieser Parametersatz muss dabei an die Aufgabe angepasst sein, die das Compoundobjekt `c` als neues Spezifikationsobjekt realisiert.

Diese nötige Zurverfügungstellung einer Parametrisierung wird in [em:] durch die Spezifikationsvariablen-Liste der Compoundobjekte realisiert. Es ist möglich, jeden Identifikator, der in `f` für eine Spezifikationsvariable oder ein Spezifikationsattribut vorkommt, in die Spezifikationsvariablen-Liste des Compoundobjektes `c` als lokale Spezifikationsvariable aufzunehmen. Für diese lokalen Spezifikationsvariablen wird dann während des Einkopierens der internen Formatierung `f` in die Designstruktur für Wertberechnungen nicht mehr, wie oben vorgestellt, der globale Spezifikationsvariablenfluss verwendet, sondern ein durch das Compoundobjekt selbst verwalteter lokaler Spezifikationsvariablenfluss, der nur innerhalb des Compoundobjektes gültig ist.

Die in diese Liste aufgenommenen Spezifikationsvariablen und die darauf beruhenden Auswirkungen auf den Spezifikationsvariablenfluss dienen somit einerseits als Individualisierungsmöglichkeit für die interne Formatierung `f`, die durch das Compoundobjekt `c` gekapselt wird: Es kann für jede der Spezifikationsvariablen in dieser Liste ein benutzerdefinierbarer Wert angegeben werden, der mit Hilfe eines Compoundobjekt-lokalen Spezifikationsvariablenflusses zur Setzung der Werte von Spezifikationsvariablen und Spezifikationsattributen verwendet wird. Andererseits dienen die in die Spezifikationsvariablen-Liste aufgenommenen Spezifikationsvariablen aber auch als Verursacher eines Compoundobjekt-internen Spezifikationsvariablenflusses, der die Auswirkungen von Manipulationen auf den lokalen Spezifikationsvariablen des Compoundobjektes auf die interne Formatierung `f` beschränkt.

Folgende Beschreibung stellt die Auswirkung des Compoundobjektes auf seinen internen Spezifikationsvariablenfluss detailliert dar:

Betreten eines Compoundobjektes: Für jede lokale Spezifikationsvariable des Compoundobjektes kann wie üblich ein [em:]-Ausdruck zur Berechnung eines Initialisierungswertes angegeben werden. Die Berechnung hierzu geschieht beim Betreten des Compoundobjektes mit Hilfe des Spezifikationsvariablenflusses. Der Spezifikationsvariablenfluss enthält nun lokal im Compoundobjekt für die Spezifikationsvariablen, die in `c` eingeführt wurden, die beim Betreten neu berechneten Werte. Das Compoundobjekt verschattet die alten Werte der in ihm neudeklarierten Spezifikationsvariablen.

Im Compoundobjekt: Dieser beim Betreten des Compoundobjektes manipulierte Spezifikationsvariablenfluss kann nun innerhalb des Compoundobjektes zur Initialisierung der Spezifikationsobjekte in `s` solange verwendet werden, bis er aufgrund eines in `s` vorkommenden Konnektors das Compoundobjekt verlässt. Der lokale Spezifikationsvariablenfluss kann dabei von den in `s` vorkommenden Auswertungsobjekten manipuliert werden.

Fortfahren aus dem Compoundobjekt: An dieser Stelle wird der Spezifikationsvariablenfluss für die lokalen Spezifikationsvariablen des Compoundobjektes wieder auf den Zustand gesetzt, den er vor dem Betreten des Compoundobjektes hatte. Für die anderen globalen Spezifikationsvariablen des Spezifikationsvariablenflusses werden keine Maßnahmen getroffen.

Rückkehren in das Compoundobjekt: Der Spezifikationsvariablenfluss wird an dieser Stelle für die lokalen Spezifikationsvariablen des Compoundobjektes auf den Zustand gesetzt, den er vor dem Fortfahren aus dem Compoundobjekt hatte.

Im Compoundobjekt: Wie schon bei der ersten Strecke im Compoundobjekt kann der bei der Rückkehr modifizierte Spezifikationsvariablenfluss zur Wertsetzung und Manipulation wie üblich verwendet werden.

Verlassen des Compoundobjektes: An dieser Stelle wird der Spezifikationsvariablenfluss für die lokalen Spezifikationsvariablen in den Zustand gesetzt, den er vor dem Rückkehren in das Compoundobjekt hatte.

Die Wirkung dieser Manipulation des Spezifikationsvariablenflusses durch ein Compoundobjekt auf die lokalen Spezifikationsvariablen des Compoundobjektes wird schematisch noch einmal in Abbildung 4.27 verdeutlicht.

4.7.2 Konnektoren in Compoundobjekten

Die bisherige Beschreibung der Compoundobjekte nahm implizit an, dass in der internen Formatierung eines Compoundobjektes nur ein Konnektor enthalten ist. Durch diese Annahme ergab sich bei den bisherigen Erläuterungen, dass der Spezifikationsvariablenfluss das Compoundobjekt nur einmal verläßt, um dann auch nur einmal zurückzukehren. Dies ist jedoch für den praxisrelevanten Einsatz der Compoundobjekte ein stark einschränkender Faktor.

Ein Beispiel, das dies unmittelbar belegt, ist die hierarchische Zusammenfassung eines `Window` und eines `Vertical` Layoutobjektes zu einem Compoundobjekt, wobei das `Vertical` Layoutobjekt zwei verschiedene Layoutflüsse akzeptieren soll (die z.B. das Inhaltsverzeichnis und den normalen Textteil eines Dokumentes transportieren). In diesem Fall ist es nötig, dass das `Vertical` Layoutobjekt zwei Konnektoren als Kinder enthält, die die entsprechenden Layoutflüsse akzeptieren. Das sich somit insgesamt ergebende Compoundobjekt hat zwei Eingänge und bewirkt, dass der Spezifikationsvariablenfluss aus dem Compoundobjekt zweimal fortfährt und entsprechend auch zweimal wieder betritt.

Es ist darauf hinzuweisen, dass das in 4.7.1 vorgestellte Modell des Spezifikationsvariablenflusses in Compoundobjekten diese erweiterte Struktur der internen Formatierungen in Compoundobjekten problemlos beschreiben kann. Als wichtigstes Ergebnis ergibt sich, dass der globale Spezifikationsvariablenfluss für jeden Konnektor mit Hilfe einer Konstruktion (siehe Abbildung 4.28) bestehend aus zwei grauen Boxen von den lokalen Manipulationen in der internen Formatierung abgeschottet wird (siehe Abbildung 4.28).

Compoundobjekte sind somit die ersten Spezifikationsobjekte, die mehr als einen Eingang haben können. An der Eigenschaft, dass sie wie alle anderen Spezifikationsobjekte entweder einen oder keinen Ausgang haben, ändert sich jedoch nichts.

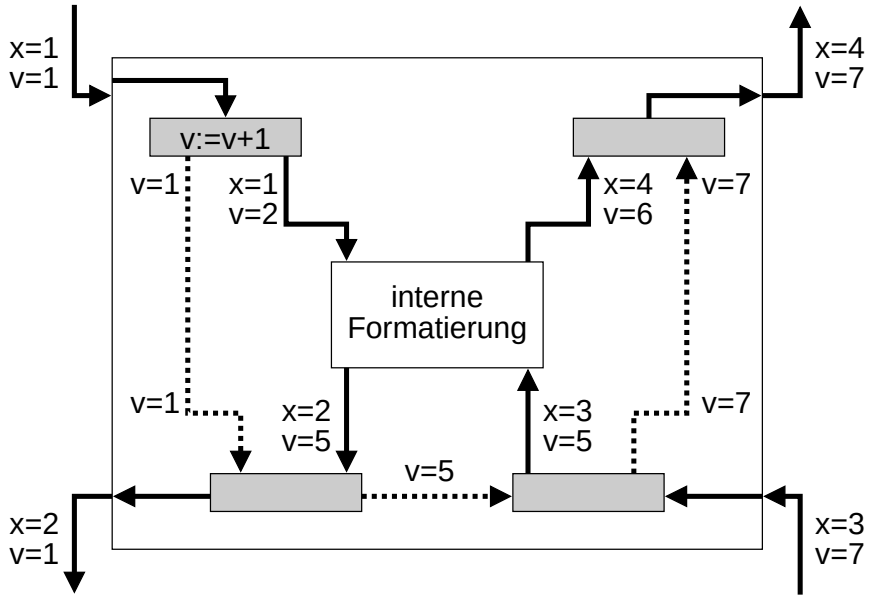

Abbildung 4.27: Lokaler Spezifikationsvariablenfluss in einem Compoundobjekt (die Spezifikationsvariable `x` ist global): Beim Betreten wird die lokale Spezifikationsvariable `v` auf den Wert `2` initialisiert und dann in der internen Formatierung auf den Wert `5` gesetzt. Beim Verlassen wird die interne Spezifikationsvariable `v` gesichert und der Wert für `v` wieder auf `1` gesetzt (mit Hilfe des linken gestrichelten Spezifikationsvariablenflusses). Bei der Rückkehr wird `v` wieder auf den gesicherten Wert `5` zurückgesetzt und dann in der internen Formatierung auf `6` geändert. Beim Verlassen wird die lokale Spezifikationsvariable `v` mit Hilfe des rechten gestrichelten Spezifikationsvariablenflusses wieder auf den beim Rückkehren gültig gewesenen Wert `7` gesetzt.

4.7.3 Zusammenfassung

Die Einführung der Compoundobjekte in [em:] bringt eine Reihe von Vorteilen mit sich, die unmittelbar auf der Hand liegen. Zunächst einmal beseitigen sie die unangenehmen Folgen der Einfachheits- und Minimalitätsforderung an die in einer Layoutobjekt-Basismenge enthaltenenen Layoutobjekte – es kann von nun an in [em:] auch komplexe Spezifikationsobjekte geben, die basierend auf der Funktionalität anderer Spezifikationsobjekte zusammengesetzt sind. Dies ermöglicht unmittelbar ein schnelleres Arbeiten bei der Erstellung von Designspezifikationen. Auch steigt die Übersichtlichkeit von mit Compoundobjekten erstellten Designspezifikationen aufgrund der kompakteren Form der in Designspezifikationen enthaltenen Designregeln. Weiterhin ermöglichen Compoundobjekte die konsistente Wiederverwendung von Spezifikationsobjekt-Kombinationen in verschiedenen Designregeln ohne das Problem zu erzeugen, bei einer nötigen Änderung alle Verwendungsstellen anzupassen; wird die Definition eines Compoundobjektes geändert, wird automatisch bei allen Zugriffen auf das Compoundobjekt konsistent die neue Definition verwendet.

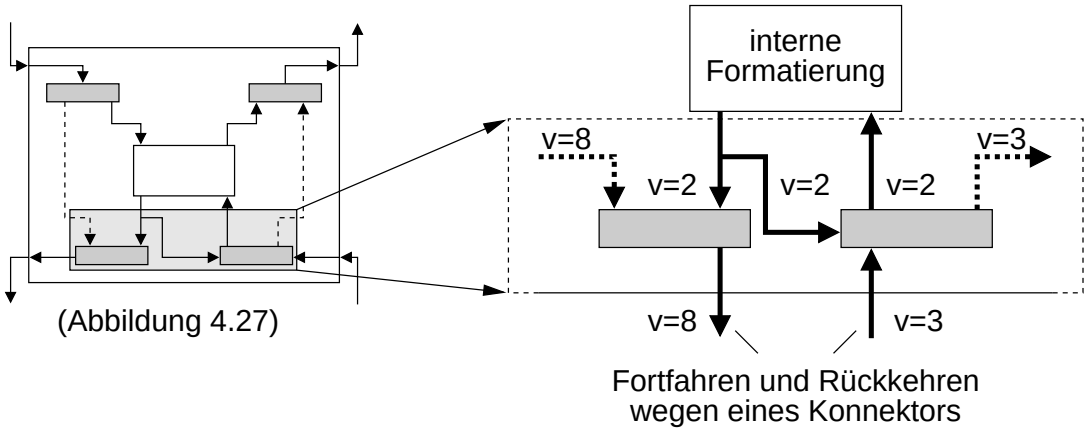

Abbildung 4.28: Ausschnitt aus einem lokalen Spezifikationsvariablenfluss in einem Compoundobjekt mit mehreren Konnektoren: Das durch einen Konnektor verursachte Fortsetzen des Spezifikationsvariablenflusses aus dem Compoundobjekt sowie die entsprechende Rückkehr wird mit Hilfe der dargestellten Konstruktion, die direkt dem unteren Teil aus Abildung 4.27 abgeleitet ist, realisiert. Für jeden in einem Compoundobjekt vorkommenden Konnektor wird sie einmal verwendet. Beim Fortsetzen wird mit Hilfe des linken gestrichelten Spezifikationsvariablenflusses der ursprüngliche Wert von v wieder hergestellt und beim Rückkehren v aufgrund einer Sicherung erneut auf seinen alten internen Wert gesetzt.

4.8 Erweiterungen

In diesem Abschnitt werden die bisherigen Konstrukte der Designspezifikationsmethodik [em:] zunächst um Möglichkeiten erweitert, die bestimmte Aspekte einer Designspezifikation durch Anreicherungsmöglichkeiten des logischen Baumes vereinfachen und Redundanz vermeiden helfen. Im Anschluß daran wird ein Konstrukt vorgestellt, mit dessen Hilfe die Überführung eines logisch ausgezeichneten Dokumentes in ein logisch ausgezeichnetes Dokument beschrieben werden kann. Abschließend werden noch einige *kleine* Hilfsobjekte vorgestellt, die den Funktionsumfang einer in [em:] erstellbaren Designspezifikation abrunden.

4.8.1 Anreicherung eines logischen Baumes

Im Rahmen einer [em:] Designspezifikation können mit Hilfe des Spezifikationsvariablenflusses Werte für Spezifikationsvariablen berechnet werden, die für die Realisierung der erweiterten Aufgaben der Dokumentformatierer (z.B. automatische Überschriftennummerierung) benötigt werden. Dieser Werte werden im Laufe einer Designapplikation mit Hilfe einer Menge von geeigneten Designregeln, die durch einen Designer aufgestellt werden, berechnet und ändern sich daher in ihrem Verlauf in einer vom Designer vorgegebenen Weise.

Soll beispielsweise ein abstraktes Dokument ohne eine eigene explizite Nummerierung seiner Bestandteile[38] (wie z.B. der Überschriften) in eine Designstruktur überführt werden, die

[38] Derartige Dokumente sind ohne Systemunterstützung leichter als Dokumente mit einer expliziten Nummerie-

aufgrund der Verwendung von zwei Views ein Inhaltsverzeichnis und einen normalen Textteil beinhaltet, so müssen im Besonderen die Überschriften des Dokumentes zweifach verarbeitet werden: Einmal für das Inhaltsverzeichnis und einmal für den Textteil. Für eine konsistente Nummerierung der Überschriften in beiden Dokumentteilen muss dabei mit Hilfe von Designregeln zweimal diese Nummerierung im Spezifikationsvariablenfluss erzeugt werden – und dies mit Hilfe von verschiedenen Designregeln für die zwei zu erzeugenden Views.

Diese Situation führt zu unnötiger Redundanz in einer Designspezifikation, die bei Änderungen schnell Konsistenzprobleme nach sich zieht. Erinnert man sich der Ursache für diese Situation, der fehlenden expliziten Nummerierung der logischen Objekte des abstrakten Dokumentes, so kann man aus ihr unmittelbar eine elegante Lösung für diese Situation ableiten: Die Einführung einer expliziten Nummerierung für die logischen Objekte im gegebenen logisch ausgezeichneten Dokument, auf die dann jederzeit im Rahmen einer Designapplikation zugegriffen werden kann.

Das soeben beschriebene Vorgehen führt zu einer *Anreicherung* eines logisch ausgezeichneten Dokumentes um Bestandteile, die in ihm nur implizit vorhanden sind, mit seiner Hilfe im Rahmen einer Designapplikation aber explizit berechnet werden können. Man erreicht damit eine Situation, in der das Dokument all diejenigen Informationen direkt bereitstellt, die man zu seiner *einfachen* Bearbeitung benötigt. Die logischen Objekte des Dokumentes dienen dabei umgangssprachlich bezeichnet als *Aufhängepunkte* für *Hilfsattribute*, deren Wirkungsbereich sich durch den normalen logischen Fluss ergibt.

In [em:] wird dies mit Hilfe eines neuen Spezifikationsobjektes erreicht, das `SetLogAtt` (Set Logical Attribute) heißt. Es kann in einer Designregel verwendet werden, um im logischen Objekt, auf das die Designregel angewendet wird, neue logische Attribute mit einem bestimmten Wert anzulegen. Dazu verfügt das `SetLogAtt` Spezifikationsobjekt über eine Spezifikationsvariablen-Liste, in die beliebige Spezifikationsvariablen aller in [em:] verfügbaren Sorten[39] aufgenommen werden können. Bei der Auswertung der `SetLogAtt` Spezifikationsobjekte werden mit Hilfe des Spezifikationsvariablenflusses wie üblich die Werte für die Spezifikationsvariablen berechnet; sie werden dann dementsprechend gesetzt in Form von logischen Attributen dem logischen Objekten zugeordnet, für das die Designregel angewendet wird. Diese Werte sind von da an im noch folgenden logischen Fluss wie gewöhnliche logische Attribute in den logischen Objekten vorhanden[40].

4.8.2 Die Überführung logisch ausgezeichneter Dokumente in strukturierte Datenstrukturen

Das Ziel bei der bisherigen Entwicklung der Designspezifikationsmethodik [em:] war die Entwicklung eines Mechanismus, mit dessen Hilfe die Überführung eines logisch ausgezeichneten

rung zu erstellen, da bereits beim Einfügen einer neuen Überschrift aufwendige Aktionen zur Aufrechterhaltung der korrekten Nummerierung nötig werden.

[39] Die bisher in einem logischen Objekt zugelassenen logischen Attribute mußten von der Sorte `String` sein.

[40] Die einzige Einschränkung dieser berechneten logischen Attribute ist ihre Nichtverwendbarkeit in Umgebungsprädikaten.

Dokumentes in eine Designstruktur beschrieben werden kann. Diese Designstruktur legt statisch einen Formatierprozess fest, für den bereits alle statisch berechenbaren Aufgaben der Dokumentformatierer (z.B. Nummerierungen und Erstellung von Verzeichnissen) berechnet sind. Abstrakt und aus einer gewissen Distanz heraus betrachtet stellt eine Designstruktur also nichts anderes als ein neues Dokument dar, das bezüglich gewisser Kriterien aufgrund eines gegebenen Ausgangsdokumentes entwickelt wurde.

Mit dieser Einsicht bietet es sich an, die Designspezifikationsmethodik [em:] zu Erweitern und zu mehr als nur der Überführung von logisch ausgezeichneten Dokumenten in Designstrukturen zu nutzen. Mit dieser Auffassung ist es dann zukünftig möglich, einen typischen Aufgabenbereich abzudecken, der in letzter Zeit immer wichtiger wird: Die Überführung logisch ausgezeichneter Dokumente in beliebige andere strukturierte Datenstrukturen. Bezüglich der Ziel-Datenstruktur sind dabei vom Aufgabengebiet her zwei prinzipielle Bereiche unterscheidbar: Logisch ausgezeichnete Dokumente und Dokumentformatierer-Eingabesprachen.

Überführung in logisch ausgezeichnete Dokumente

Bei der Überführung eines logisch ausgezeichneten Dokumentes in ein logisch ausgezeichnetes Dokument kann man zunächst unterscheiden, ob für das Zieldokument eine DTD vorgegeben ist oder nicht. Im ersten Fall muss nach einer erfolgten Transformation unbedingt geprüft werden, ob das neu erzeugte Dokument zu der vorgegebenen Ziel-DTD konform ist – eine Eigenschaft, die in einem beliebigen Konvertierungssystem nicht einfach zu garantieren ist.

Grundsätzlich stehen in beiden Fällen zwei Möglichkeiten der Verarbeitung offen, die beide in [em:] unterstützt werden: Dies ist zum einen die *einfache* Konvertierung eines logisch ausgezeichneten Dokumentes einer gegebenen Auszeichnungssprache in eine andere Auszeichnungssprache, wobei im Wesentlichen nur die Namen der logischen Objekte geändert werden. Zum anderen ist es die Anreicherung und der strukturelle Umbau eines Dokumentes, wobei die Ziel-Auszeichnungssprache ebenfalls unabhängig von der Quell-Auszeichnungssprache sein kann.

Gelöst wird die Aufgabe der Überführung von logisch ausgezeichneten Dokumenten in logisch ausgezeichnete Dokumente in [em:] mit Hilfe zweier neuer Spezifikationsobjekte, dem `Tagger` und einem modifizierten `Glyphs` Spezifikationsobjekt[41]. Mit Hilfe ihrer Verwendung in Designregeln kann ein strukturiertes Dokument erzeugt werden, das analog zu den in 2.3.1 beschriebenen Konzepten aufgebaut ist. Die `Tagger` Spezifikationsobjekte dienen dabei zur Beschreibung der logischen Auszeichnung, die modifizierten `Glyphs` Spezifikationsobjekte zur Realisierung des Textes eines Dokumentes. Eine ausschließlich aus diesen zwei Objekten bestehende Struktur ist dann nicht mehr als ein Layout-Pipeline-System zu betrachten, sondern vielmehr im übertragenen Sinne als ein logischer Baum.

Da das `Tagger` Spezifikationsobjekt logische Objekte modelliert wäre es zunächst ausreichend, wenn es ein Spezifikationsattribut `name` hätte, mit dem für das `Tagger` Spezifikationsobjekt der Name des durch ihn realisierten logischen Objektes gesetzt werden könnte. Aufgrund der Tatsache, dass in SGML und XML Dokumentbestandteile mit Hilfe eines Start- und eines

[41] Dieses modifizierte `Glyphs` Spezifikationsobjekt verfügt über ein Hilfsattribut, mit dessen Hilfe gesteuert werden kann, ob es als Layoutobjekt agieren, oder lediglich für seine neue Aufgabe Text repräsentieren soll.

End-Tags geklammert werden können, wird jedoch auch beim `Tagger` Spezifikationsobjekt dieser Weg gewählt und es können (vereinfacht dargestellt) mit Hilfe von zwei Spezifikationsattributen zwei Namen (`pre` für den Start- und für `post` den End-Tag) angegeben werden – schließlich ist es das Ziel, einen mit Hilfe von `Tagger` und modifizierten `Glyphs` Spezifikationsobjekten beschriebenen logischen Baum linearisiert in Textform auszugeben. Weiterhin verfügt das `Tagger` Spezifikationsobjekt über eine Spezifikationsvariablen-Liste (für deren Elemente wie üblich Werte berechnet werden können), die ebenfalls (im Start- und/oder End-Tag) ausgegeben werden kann. Das modifizierte `Glyphs` Spezifikationsobjekt verfügt nur über ein `input` Spezifikationsattribut, das die Zeichenkette enthält, die es repräsentiert.

In Abbildung 4.29 ist ein Beispiel für eine Überführung eines logischen Dokumentes, bei der folgende Designspezifikation verwendet wird, dargestellt. Es handelt sich dabei um eine einfache Überführung, bei der lediglich die Namen der logischen Objekte des Eingabedokumentes umbenannt werden; die Struktur des Eingabedokumentes bleibt erhalten.

```
Dokument  ←  (Tagger
             pre:="Memo"
             post:="Memo"
             (Konnektor
             )Konnektor
             )Tagger

text      ←  (Glyphs
             input:=@input
             )Glyphs
```

```
Paragraph ←  (Tagger
             pre:="P"
             post:="P"
             (Konnektor
             )Konnektor
             )Tagger
```

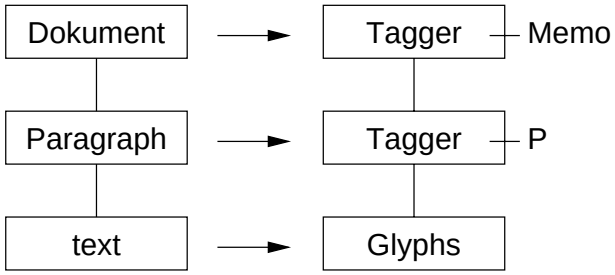

Abbildung 4.29: Überführung eines gegebenen logisch ausgezeichneten Dokumentes (links) in ein neues logisch ausgezeichnetes Dokument (rechts) unter Verwendung einer [em:] Designspezifikation.

Obgleich der Tatsache, dass ein mit Hilfe der zwei Spezifikationsobjekte `Tagger` und `Glyphs` beschriebener logischer Baum eben nur diese zwei Spezifikationsobjekte enthält, können und werden die zum Aufbau dieses logischen Baumes verwendeten Designregeln auch weitere Spezifikationsobjekte beinhalten: Auswertungsobjekte und Konnektoren. Sie werden dabei insbesondere zur Realisierung der erweiterten Aufgaben von Dokumentformatierern verwendet.

Überführung in Dokumentformatierer-Eingabesprachen

Neben der Überführung eines logisch ausgezeichneten Dokumentes in ein logisch ausgezeichnetes Dokument ist es mit Hilfe des `Tagger` und des `Glyphs` Spezifikationsobjektes weiterhin möglich, logisch ausgezeichnete Dokumente in eine Eingabe für bereits existierende Dokumentformatierer zu überführen. Es ist dabei jedoch zu betonen, dass die zu erstellende Dokumentformatierer-Eingabe gewissen syntaktischen und semantischen Regeln unterliegt, die eingehalten werden müssen; ein Schwachpunkt, der in der [em:] Designspezifikationsmethodik mit Einführung der Layoutobjekte ausgeräumt wurde. Die korrekte Erstellung einer Eingabe für existierende Dokumentformatierer ist deshalb von [em:] aus nicht zu garantieren, da in [em:] im Rahmen einer Designapplikation basierend auf benutzervorgegebenen Designregeln beliebige Strukturen erzeugt werden können – die Verantwortung liegt somit einzig in der Hand der Ersteller einer Designspezifikation.

Ein erstes typisches Beispiel für einen Dokumentformatierer, für den mit Hilfe von [em:] ausgehend von einem logisch ausgezeichneten Dokument eine Eingabe erstellt werden kann, ist ein beliebiger derzeit verfügbarer HTML-Browser. Die zu verwendende Syntax ist in diesem Falle über die standardisierte Sprache HTML festgelegt. Diese ist zwar vom Konzept her nichts anderes als eine logische Auszeichnungssprache, sie wird jedoch dazu missbraucht, Hierarchien von Formatierfunktionen zu beschreiben und ist somit eine Dokumentformatierer-Eingabesprache. Dieses Beispiel der Überführung eines logisch ausgezeichneten Dokumentes nach HTML zeigt sehr deutlich die enge Verwandschaft der Aufgabe der Überführung eines logisch ausgezeichneten Dokumentes in ein logisch ausgezeichnetes Dokument mit der hier vorgestellten Überführung in Dokumentformatierer Eingaben.

An dieser Stelle werden noch kurz zwei weitere naheliegende Zielsprachen für eine mögliche Konvertierung aufgeführt: LaTeX (siehe z.B. [36]) und RTF (siehe [40]). Beide Sprachen sind, wie schon bereits HTML, strukturierte Dokumentformatierer-Eingabesprachen, die mit Hilfe von Tags die Anwendungsbereiche von Formatierfunktionen kennzeichnen, die auf einen gegebenen Text angewendet werden sollen.

4.8.3 Hilfsobjekte für Designspezifikationen

Das Verwerfen von Teilen einer Designstruktur

Designspezifikationen werden in [em:] ebenso wie in anderen Designspezifikationssystemen inkrementell entwickelt werden. Im Rahmen einer derartigen Entwicklung wird dabei der Interessensschwerpunkt eines Designers immer auf den aktuell entwickelten Aspekten der Designspezifikation liegen. Dabei kann es passieren, dass gewisse Teile eines Dokumentes, deren Design schon vollständig spezifiziert ist, nicht mehr von Interesse sind und deshalb bei einer Preview-Möglichkeit für das Ergebnis einer Designapplikation auf ein logisch ausgezeichnetes Dokument nicht mehr realisiert werden sollen.

Aus diesem Grund wird in [em:] ein neues Layoutobjekt namens `Trash` eingeführt, das, wie das `Window` Layoutobjekt, zur Klasse der Zielobjekte gehört. Mit seiner Hilfe wird deshalb

spezifiziert, dass die es erreichenden Layoutbestandteile im Layoutfluss nicht weitergegeben werden. Weiterhin spezifiziert es, dass diese Layoutbestandteile einfach *verschluckt* werden, wie man es auch von einem Abfalleimer (Trash) erwartet. Es ist zu betonen, dass das `Trash` Layoutobjekt trotzdem wie jedes Layoutobjekt die Auswertung aller seiner Kinder anstößt und somit alle Manipulationen des Spezifikationsvariablenflusses während ihrer Auswertung ausgeführt werden. Somit wird gewährleistet, dass alle durch die Auswertung des verworfenen Teilbaumes entstehenden Auswirkungen auf die erweiterten Aufgaben der Dokumentformatierer (z.B. Nummerierungen von Dokumentbestandteilen) trotzdem ausgeführt werden. Analoges gilt für die `Beamer` Spezifikationsobjekte.

Externalisierung von Designstrukturen

Mit Hilfe der bisher eingeführten Konstrukte der [em:] Methodik können Designstrukturen und, wie in 4.8.2 gezeigt wurde, weitere hierarchische Datenstrukturen aufgebaut werden. Es gibt aber bisher in [em:] keine Möglichkeit, eine persistente Speicherung dieser Strukturen zu spezifizieren. Aus diesem Grund wird in [em:] das `Protocol` Spezifikationsobjekt eingeführt, das die Speicherung seines gesamten Unterbaumes in eine Datei spezifiziert, die mit Hilfe des Spezifikationsattributes `outputfileName` angegeben werden kann.

Aus Praxisgründen wird in [em:] zugleich ein zu dem `Protocol` Spezifikationsobjekt inverses Spezifikationsobjekt eingeführt, das `Reader` Layoutobjekt. Dieses spezifiziert die Verwendung der in einer Datei (über das Spezifikationsattribut `inputFileName` angebbar) abgelegten Designstruktur zum Einkopieren in eine Designstruktur. Das `Reader` Spezifikationsobjekt fällt somit in die Klasse der Quellobjekte.

Aufruf [em:]-externer Funktionalität

Als letzte Spezifikationshilfe steht noch ein weiteres Spezifikationsobjekt zur Verfügung, das `SysCall` Objekt. Mit seiner Hilfe kann der Aufruf einer [em:]-externen Funktionalität in einer [em:] Designspezifikation spezifiziert werden. Dazu verfügt das `SysCall` Spezifikationsobjekt über ein Spezifikationsattribut, das `shellCommand` heißt. Das dort spezifizierte Kommando wird während der Auswertung eines `SysCall` Spezifikationsobjektes im Rahmen einer Designapplikation ausgeführt. Die dabei entstehenden Auswirkungen (z.B. das Anlegen einer Datei) können bei der restlichen Ausführung der Designapplikation ausgenutzt werden (z.B. Verwendung der angelegten Datei in einem `Repeater` Spezifikationsobjekt).

4.9 Die [em:]-Ausdruckssprache

In der Designspezifikationsmethodik [em:] können an den verschiedenensten Stellen programmiersprachliche Ausdrücke zur Setzung der Werte von Spezifikationsattributen und -variablen

sowie der Berechnung von Umgebungsprädikaten verwendet werden. Diese programmiersprachlichen Ausdrücke heißen [em:]-Ausdrücke. Sie müssen, wie in allen Programmiersprachen üblich, gewissen Regeln folgen, die in diesem Fall in der [em:]-Ausdruckssprache festgelegt sind. Weiterhin werden in der [em:]-Ausdruckssprache auch Vorgaben getroffen, wie die Identifikatoren für Spezifikationsattribute bzw. -variablen sowie logische Attribute aufgebaut sein müssen und wie [em:]-Ausdrücke berechnet werden.

4.9.1 [em:]-Ausdrücke und ihre Berechnung

[em:]-Ausdrücke der [em:]-Ausdruckssprache werden in Spezifikationsobjekten zur Berechnung der Werte von Spezifikationsattributen, in Auswertungsobjekten zur Berechnung der Werte von Spezifikationsvariablen sowie bei der Berechnung von Umgebungsprädikaten verwendet.

Zur Bildung von [em:]-Ausdrücken können als elementare Bestandteile Identifikatoren gemäß 4.9.2 und die Konstanten der in der [em:]-Ausdruckssprache in 4.9.3 definierten Sorten verwendet werden. Zur Verknüpfung von ihnen stehen die in 4.9.4 definierten Funktionen und Operatoren bereit.

In einem Ausdruck der [em:]-Ausdruckssprache dürfen als Identifikatoren in Umgebungsprädikaten nur logische Attribute, bei den anderen Verwendungsmöglichkeiten der [em:]-Ausdrücke zusätzlich auch Spezifikationsvariablen verwendet werden. Spezifikationsattribute können in einem [em:]-Ausdruck nicht vorkommen, da nach Konvention die in einem [em:]-Ausdruck vorkommenden Identifikatoren, die nicht für ein logisches Attribut stehen, eine Spezifikationsvariable identifizieren[42]. Auf die Spezifikationsattribute von Spezifikationsobjekten kann, wie sich zeigen wird, somit nur schreibend zugegriffen werden.

Die Werte für die in einem [em:]-Ausdruck `expr` vorkommenden Identifikatoren während einer Berechnung von `expr` ergeben sich im Verlauf einer Designapplikation gemäß dem Verfahren, das in Abschnitt 4.4.5 (Spezifikationsvariablenfluss) und 4.4.6 (logischer Fluss) beschrieben wurde.

Zur Ausführung der Berechnung eines [em:]-Ausdrucks `expr` ist es Voraussetzung, dass alle in `expr` vorkommenden Spezifikationsvariablen und logischen Attribute deklariert sind. Ist dies nicht der Fall, so werden die Zugriffe auf die nicht-deklarierten Identifikatoren als *illegal* bezeichnet. Um zu einem fehlertoleranten Verhalten des [es:]-Systems zu gelangen, werden derartige illegale Zugriffe während der Auswertung eines [em:]-Ausdrucks folgendermaßen behandelt: Erfolgt ein illegaler Zugriff auf ein logisches Attribut, so wird der leere String als Ergebnis geliefert. In allen anderen Fällen wird jedoch ein Fehlerzustand erzeugt. Dieses Verhalten wird gewählt, da ein nichtdeklariertes logisches Attribut aufgrund einer unvorhergesehenen Anwendungsstelle einer Designregel während einer Designapplikation *entstehen* kann – eine Tatsache, die im Rahmen einer Designspezifikation vorkommen *kann*, wenngleich sie es nicht sollte. Ein Zugriff auf eine nichtdeklarierte Spezifikationsvariable hingegen ist prinzipiell

[42] Diese Festlegung ist nötig, da nach Definition 50 zu jedem Spezifikationsattribut eine Spezifikationsvariable mit gleichem Namen und gleicher Sorte eingeführt wird.

an einer beliebigen Stelle in einer erweiterten Designstruktur ein Fehler – eine Eigenschaft einer Designspezifikation, die *nicht vorkommen darf* und schon während der Designspezifikation auffallen müsste.

Neben der Verwendung eines [em:]-Ausdrucks als Umgebungsprädikat kann ein [em:]-Ausdruck auch zur Setzung des Wertes eines Spezifikationsattributes oder einer Spezifikationsvariablen verwendet werden[43]. Die Auswirkungen einer derartigen Zuweisung sind bereits ausführlich in Abschnitt 4.4.5 erklärt worden. Im Rahmen einer Zuweisung ist darauf zu achten, dass die Sorte des Spezifikationsattributes bzw. der Spezifikationsvariablen, der ein [em:]-Ausdruck `expr` zugewiesen wird, zu diesem Ausdruck `expr` *verträglich* ist (siehe hierzu 4.9.5).

Folgende Beschreibung definiert eine Zuweisung in der [em:]-Ausruckssprache. `id` ist dabei ein Platzhalter für ein Spezifikationsattribut bzw. eine Spezifikationsvariable, der der Wert des Ausdrucks `expr` zugewiesen wird. Die oben eingeführten Nebenbedingungen sind zu beachten.

Ergebnissorte	Funktion/Operation	Bedeutung
keine	$id := expr$	Zuweisung

Seiteneffekte

Die soeben vorgestellte Möglichkeit einer Zuweisung in der [em:]-Ausdruckssprache liefert kein Ergebnis. Ihre Wirkung muss jedoch trotzdem für das Auswertungssystem der [em:]-Ausdrücke in irgendeiner Form protokolliert werden, um bei folgenden Berechnungen berücksichtigt werden zu können.

Zur Lösung dieser Aufgabe wird eine Symboltabelle verwendet, die die Realisierung des Spezifikationsvariablenflusses übernimmt. Jede Zuweisung an eine Spezifikationsvariable, die in einem Auswertungsobjekt erfolgt, wird über diese Symboltabelle entsprechend der Funktionalität des Auswertungsobjektes verwaltet. Es ist anzumerken, dass auch die logischen Attribute in der Symboltabelle verwaltet werden. Die Setzung ihrer Werte erfolgt jedoch ausschließlich durch die logischen Objekte und kann somit nicht direkt durch die Nutzer beeinflusst werden.

Auch die in 4.9.4 eingeführten Funktionen und Operationen können Seiteneffekte im Auswertungssystem hervorrufen. Dies ist insbesondere dann der Fall, wenn eine Berechnung aufgrund einer Sortenunverträglichkeit oder unzulässiger mathematischer Berechnungen (Division durch `0`) nicht ausgeführt werden kann.

In folgenden im Fließtext eingebundenen Beispielen werden [em:]-Ausdrücke zur Kenntlichmachung immer in die Zeichen » und « eingeschlossen.

[43] Es ist darauf hinzuweisen, dass auf logische Attribute nur lesend zugegriffen werden kann.

4.9.2 Identifikatoren

Für die Identifikatoren (die Namen) der Spezifikationsattribute und Spezifikationsvariablen sowie den logischen Attributen gelten in [em:] die in vielen Programmiersprachen üblichen Konventionen: Ein Name ist eine beliebig lange Zeichenkette, die aus den Zeichen `A`, ..., `Z`, `a`, ..., `z`, `0`, ..., `9` und _ besteht, wobei das erste Zeichen keine Ziffer sein darf. Groß- und Kleinbuchstaben werden dabei unterschieden. Die Namen für logische Attribute müssen zusätzlich zu obiger Regel mit einem `@` beginnen.

Ein Name in der [em:]-Ausdruckssprache darf nicht gleich einem Schlüsselwort der [em:]-Ausdruckssprache sein. Als Schlüsselwörter zählen alle Namen von Funktionen, Operatoren und Längeneinheiten der [em:]-Ausdruckssprache.

4.9.3 Sorten

[em:]-Ausdrücke, Spezifikationsattribute sowie -variablen können in [em:] eine der in folgender Aufzählung eingeführten vier Sorten `Bool`, `Number`, `Length` und `String` haben, logische Attribute sind immer von der Sorte `String`, Umgebungsprädikate von der Sorte `Bool`:

Bool Wahrheitswert:

Die verfügbaren `Bool`-Konstanten sind: `true` und `false`.

Number Zahlenwert:
Es wurde bewusst auf eine Trennung in Gleitkomma- und ganze Zahlen verzichtet und stattdessen die Sorte `Number` so definiert, dass der Benutzer die Genauigkeit in Nachkommastellen angeben kann. Dies hat den Vorteil, dass jede Zahl bei der Umwandlung in einen String (die implizit über einen Cast-Mechanismus geschehen kann) automatisch mit der gewünschten Genauigkeit dargestellt wird. Intern wird ein `Number`-Wert als Gleitkommazahl (mit zusätzlicher Information über die Nachkommastellen) repräsentiert, so dass das Problem der Rundungsfehlerakkumulation bei Operationen wie der Division vermindert wird. Nach außen hin wird die Zahl jedoch nur mit der Genauigkeit der definierten Nachkommastellen repräsentiert, was sich z.B. auch bei Vergleichsoperationen bemerkbar macht. Operationen, die ganze Zahlen als Argument benötigen, ignorieren die Nachkommastellen des entsprechenden `Number`-Wertes. Die Anzahl der Nachkommastellen eines vorgegebenen `Number`-Ausdrucks ergibt sich entweder bei Konstanten direkt aus der Schreibweise oder kann durch den `:`-Operator (siehe 4.9.4) festgelegt werden.

Beispiele für `Number`-Konstanten: `123`, `3.1415`, `-7.00`

Length Längenangabe:
Die Sorte `Length` entsteht durch Kombination eines Zahlenwertes mit einer Längeneinheit. Als solche stehen Millimeter (`mm`), Zentimeter (`cm`), Zoll (`inch`) und Punkt (`pt`) zur Verfügung. Bei Operationen mit `Length`-Werten verschiedener Einheit werden die Einheiten automatisch umgerechnet.

Beispiele für `Length`-Konstanten:
`1.0 cm, 2.3 mm, 0.5 inch, 12 pt`

String Zeichenketten:
`String`-Konstanten können durch doppelte oder einfache Anführungszeichen begrenzt werden, das Begrenzungszeichen darf in der Zeichenkette nicht enthalten sein.

Beispiele für `String`-Konstanten:
`"Beispiel", "", "Sie sagte 'Hallo'.",`
`'Er sagte "Hallo".'`

4.9.4 Funktionen und Operatoren

Folgende Zusammenstellung zeigt eine Übersicht der Funktionen und Operatoren, die auf [em:]-Ausdrücken ausgeführt werden können. Dabei werden je nach Sorte verschiedene Beispiel-Spezifikationsvariablen verwendet: b_i für `Bool`, n_i für `Number`, l_i für `Length`, s_i für `String` und t_i für eine beliebige Sorte, wobei die t_i innerhalb einer Operation dieselbe Sorte haben müssen.

Ergebnissorte	Funktion/Operation	Bedeutung
`Bool`	`not` b_2 b_1 `or` b_2 b_1 `and` b_2	boolesche Operatoren
	t_1 `==` t_2 t_1 `!=` t_2 t_1 `<=` t_2 t_1 `>=` t_2 t_1 `<` t_2 t_1 `>` t_2	Vergleichsoperatoren (Strings werden nach ihrer lexikographischen Ordnung verglichen)
	s_1 `in` s_2	`true` gdw. das erste Zeichen von s_1 in s_2 vorkommt

Ergebnissorte	Funktion/Operation	Bedeutung
`Number` oder `Length` (jeweils Sorte der t_i)	t_1 `+` t_2 t_1 `-` t_2 n_1 `*` t_1 t_1 `*` n_1 t_1 `/` n_1 `-`t_1	arithmetische Operationen
	t_1`:`n_1	Ändern der Nachkommastellen von t_1 auf n_1

Ergebnissorte	Funktion/Operation	Bedeutung
`Number`	n_1 `mod` n_2	Rest einer ganzzahligen Division
	l_1 `/` l_2	Division zweier Längen

Ergebnissorte	Funktion/Operation	Bedeutung
`String`	$s_1 \sim s_2$	Konkatenation von Zeichenketten
	`toUpper(`s_1`)` `toLower(`s_1`)`	Umwandlung von Kleinbuchstaben in Großbuchstaben bzw. umgekehrt
	`Roman(`n_1`)` `roman(`n_1`)`	Darstellung von n_1 in großen bzw. kleinen römischen Ziffern

	`day()`
	`date()`
	`time()`
		liefert den aktuellen Wochentag
		liefert das aktuelle Datum
		liefert die aktuelle Uhrzeit

Ergebnissorte	Funktion/Operation	Bedeutung
beliebig (jeweils Sorte der t_i)	`min(`t_1`,` t_2`)` `max(`t_1`,` t_2`)`	Minimum bzw. Maximum bezüglich <
	`if` b_1 `then` t_1 `else` t_2	bedingter Ausdruck

4.9.5 Casting

Die Sprache der [em:]-Ausdrücke an sich ist streng typisiert; d.h., bei der Berechnung eines [em:]-(Teil-)Ausdrucks dürfen nur Sorten vorkommen, für die die anzuwendenden Funktionen und Operationen definiert sind. Es gibt hierzu jedoch zwei Ausnahmen, die aus Gründen einer Vereinfachung der [em:]-Ausdruckssprache eingeführt wurden. Im Falle eines beliebigen Eintretens dieser Fälle wird jeweils ein *automatischer Cast* durchgeführt, der ein nicht passendes Argument einer Funktion bzw. Operation in ein passendes überführt.

Die Verträglichkeit eines (Teil-)Ausdrucks ergibt sich somit wie folgt: Ein (Teil-)Ausdruck ist verträglich zu seiner Anwendungsstelle, wenn er von der richtigen Sorte ist, oder mit Hilfe eines der beiden folgenden Casts in die richtige Sorte überführt werden kann.

Number-to-String Cast Ein [em:]-Ausdruck der Sorte `Number` wird bei einer Auswertung, falls nötig, automatisch in einen Ausdruck der Sorte `String` umgewandelt. Dies vereinfacht die Syntax eines [em:]-Ausdrucks beispielsweise bei der Einbindung von Zählern in Kapitelüberschriften.

Beispiel:
Sei `nr` eine Spezifikationsvariable der Sorte `Number` mit dem Wert `4`, die in dem [em:]-Ausdruck »`"Kapitel "~nr`« verwendet wird.

Dieser liefert bei seiner Auswertung mit Hilfe der Ausführung eines automatischen Casts, der die Spezifikationsvariable `nr` mit dem `Number`-Wert »`4`« in den `String`-Wert »`"4"`« überführt, als Ergebnis »`"Kapitel 4"`«.

String-to-Number Cast Funktional invers zu obigem Cast wird hier ein [em:]-Ausdruck der Sorte `String` bei einer Auswertung, falls nötig, automatisch in einen Ausdruck der Sorte `Number` umgewandelt. Dies ist z.B. dann von Nutzen, wenn ein logisches Attribut, das nach Definition immer von der Sorte `String` ist, in einer `Number`-Berechnung verwendet werden soll. Dieser Cast liefert `0` als Ergebnis, wenn keine *gültige* Zahl zu Beginn der Zeichenkette steht.

Beispiel:
Sei das logische Attribut `@id` mit dem `String`-Wert »`"8"`« belegt. Zur Berechnung des [em:]-Ausdrucks »`@id+1`« wird ein automatischer Cast angewendet, der für den Teilausdruck »`@id`« der Sorte `String` den `Number`-Ausdruck »`8`« liefert, und damit insgesamt die Berechnung des Ausdrucks zu »`9`« ermöglicht.

Kapitel 5

Kontextabhängige Verarbeitung von Dokumenten

Anerkanntermaßen ist man im Bereich der Verarbeitung von logisch ausgezeichneten Dokumenten übereingekommen, dass bei der Verwendung von regelbasierten Ansätzen die Auswahl der anzuwendenden Designregel über boolesche Prädikate gesteuert werden muss [7] [53] [54] [55] [57] [30]. In Bezug auf den Aufbau der benötigten booleschen Prädikate, ihre Ausdrucksstärke sowie die Vorgehensweise zur ihrer Erstellung in konkreten Systemen sind bis jetzt jedoch noch keine einheitlichen Tendenzen zu erkennen[1]. Die Anforderungen sind je nach Anwendungsgebiet, in der die Verarbeitung des Dokumentes stattfindet, zu unterschiedlich, als dass man einen allgemein verwendbaren Standard hätte entwickeln wollen bzw. können.

Aus diesem Grund wird in diesem Kapitel zunächst eine Untersuchung und Klassifizierung der möglichen Faktoren, die zum Verarbeitungskontext eines Dokumentes beitragen können, vorgenommen. Im Anschluß daran werden die strukturellen Kontexte, die ausschließlich auf der logischen Struktur von logisch ausgezeichneten Dokumenten aufsetzen und für diese Arbeit von besonderer Bedeutung sind, untersucht. Aufbauend auf dieser Untersuchung wird das für die Umgebungsprädikate von [em:] entwickelte und verwendete T-Kontextmodell vorgestellt und eine Methodik aufgezeigt, mit deren Hilfe Umgebungsprädikate gemäß dem T-Kontextmodell nutzerfreundlich aufgestellt werden können.

5.1 Klassifizierung von Kontexten

Laut [21] ist der Kontext eines Gegenstandes die enge Verknüpfung von ihm mit anderen Gegenständen, der inhaltliche Zusammenhang zwischen ihnen, sowie in Bezug auf einen Text auch der ihn umgebende Text. Diese allgemeine Aussage weist in Bezug auf den Kontextbegriff, wie er bei der Bearbeitung von logisch ausgezeichneten Dokumenten benötigt wird, bereits in die

[1] Mit dem CSS3 Modul W3C selectors wird jedoch aktuell (August 1999) ein erster, wenn auch inhaltlich stark eingeschränkter, Schritt unternommen, eine allgemein anerkannte Basis bezüglich des Aufbaus und der Ausdrucksstärke zu schaffen.

korrekte Richtung. Hier sind die oben erwähnten Gegenstände die logischen Objekten eines logisch ausgezeichneten Dokumentes sowie die in ihnen beinhalteten kleineren Objekte (wie z.B. der Text und somit die einzelnen Sätze, Wörter und Buchstaben eines Dokumentes), die nicht explizit als einzelne logische Objekte ausgezeichnet sind.

Der Kontext dieser soeben aufgeführten Dokumentbestandteile ergibt sich dann zum einen aus den restlichen Dokumentbestandteilen ihres logisch ausgezeichneten Dokumentes sowie zum anderen aus der großen Menge an weiteren Faktoren, die auf ein Dokument während dessen Verarbeitung und Betrachtung durch einen Nutzer einwirken können. In Abbildung 5.1 ist ein schematischer Überblick über eine mögliche Klassifizierung interessanter Kontexte, die im Zusammenhang mit der Verarbeitung von logisch ausgezeichneten Dokumenten auftreten können, aufgezeigt.

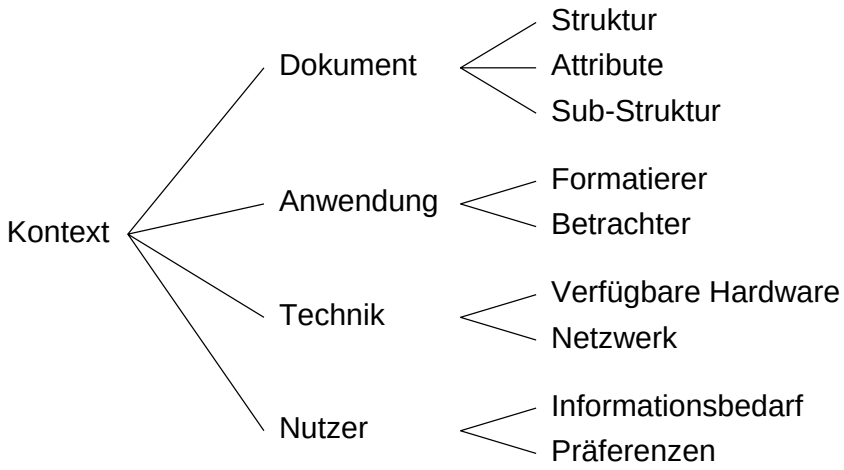

Abbildung 5.1: Kontextarten für ein logisch ausgezeichnetes Dokument: Von links nach rechts wird der Kontextbegriff spezialisiert.

Die verschiedenen Kontextarten, die in der in Abbildung 5.1 vorgenommenen Klassifizierung unterschieden werden, können sich allesamt auf die Verarbeitung eines Dokumentes (die Überführung von einem abstrakten in ein konkretes Dokument) auswirken und werden im Folgenden kurz erläutert:

Nutzer: Die Nutzer bringen als Konsumenten eines Dokumentes die verschiedensten Anforderungen an ein Dokument mit sich. Dies sind zum einen persönliche *Präferenzen*, die sich z.B. aufgrund der Kurzsichtigkeit oder Farbenblindheit eines Nutzers ergeben.
Zum anderen kann sich aber auch der *Informationsbedarf* eines Nutzers auf die Verarbeitung eines Dokumentes auswirken. Enthält ein Dokument z.B. die Gebrauchsanleitung für verschiedene Varianten eines Produktes, so hat ein Käufer kein Interesse daran, die für ihn irrelevanten Teile lesen zu müssen – sie sollten für ihn ausgeblendet werden.

Technik: Technische Kontexte, die sich bei der Verarbeitung eines Dokumentes ergeben, können z.B. von dem *Netzwerk* herrühren, über das ein Dokument an einen Nut-

zer zur Verfügung gestellt wird. Je nach verfügbarer Bandbreite können im Rahmen der Lieferung eines Dokumentes an einen Nutzer verschiedene Graphiken (hohe/niedrige Qualität) in dieses eingebunden werden.
Die verfügbare *Hardware* kann ebenfalls Auswirkungen auf das einem Nutzer zur Verfügung gestellte konkrete Dokument haben: Je nach Graphikkarte (Farbe/SW), Monitor (Größe) oder Eingabemedium (Touch-Screen, Maus, Tastatur, Stift) können verschiedene konkrete Dokumente nötig sein.

Anwendung: Der Kontext, der sich in einer Anwendung ergibt, die ein Dokument verarbeitet, kann sich z.B. zu dem Zeitpunkt ergeben, an dem ein konkretes Dokument von einem Nutzer *betrachtet* wird. Typisches Beispiel hierfür ist die Tatsache, dass ein Nutzer schon mehrere Verweise, die in einem Dokument vorkommen, besucht hat – diese sollen dann farblich von den noch nicht besuchten Verweisen abgehoben werden.
Eine andere Art von Kontext ergibt sich bei der Verarbeitung eines Dokumentes mit Hilfe eines *Formatierers*. Beispiele für Kontexte, die hier auftreten können, sind z.B. die besondere Formatierung des Textes in der ersten Zeile eines Absatzes, ein besonders zu setzender Einzug für einen Absatz mit nur einer Zeile oder auch die unterschiedliche Verarbeitung von Inhalt, je nachdem ob er auf einer linken oder rechten Seite gesetzt wird.

Dokument: In einigen konkreten Dokumenten ist es erforderlich, den ersten Buchstaben oder das erste Wort eines Absatzes in einer besonderen Schrift zu setzen oder bestimmte Klassen von Zeichen des abstrakten Dokumentes je nach ihrer Vernestung (z.B. Anführungszeichen) unterschiedlich zu behandeln; der Kontext, unter dem diese besondere Verarbeitung nötig wird, ergibt sich aus der *Sub-Struktur*[2] eines logisch ausgezeichneten Dokumentes.
Auch die *logischen Attribute* der logischen Objekte eines Dokumentes können einen Kontext zur Verarbeitung des Dokumentes vorgeben: Typische Beispiele hierfür sind das logische Attribut `ID` (siehe [32]) und das logische Attribut `CLASS` (siehe [53]), aufgrund derer ein einzelnes bzw. eine Gruppe von logischen Objekten eine spezielle Verarbeitung erhalten können.
Der letzte oben aufgeführte Grund, für den sich aufgrund eines Dokumentes für die Verarbeitung eines Dokumentes ein Kontext ergeben kann, ist die *logische Struktur* des Dokumentes. Dieser Punkt wird im Folgenden genauer untersucht werden.

Die oben vorgestellten vier Hauptklassen von Kontexten treten zu verschiedenen Zeitpunkten bei der Bearbeitung eines logisch ausgezeichneten Dokumentes auf dem Weg vom abstrakten zum konkreten Dokument auf. Ein mögliches (wenn auch nicht vollständiges) Szenario hierfür ist in Abbildung 5.2 dargestellt. Ein gegebenes abstraktes Dokument wird dort aufgrund verschiedener Kontexte, die bei der Ausführung der Verarbeitungsstufen des Dokumentes gegeben sind, schrittweise verarbeitet. Die Kontexte spiegeln dabei jeweils die Anforderungen

[2] Die Struktur, die unterhalb der logischen Auszeichnung eines Dokumentes angesiedelt ist – sie ist somit nicht *explizit* in einem Dokument durch einen Ersteller vorgegeben.

wieder, die in den einzelnen Stufen an die Verarbeitung des Dokumentes gestellt werden, um am Ende ein an eine Aufgabenstellung angepasstes konkretes Dokument zu erhalten.

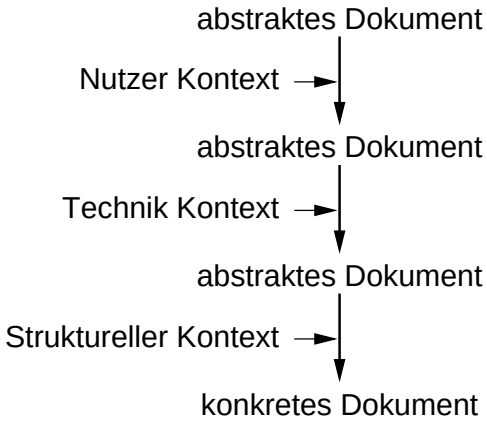

Abbildung 5.2: Verwendete Kontexte bei der schrittweisen Verarbeitung eines Dokumentes

Auch wenn das in Abildung 5.2 dargestellte Beispiel nicht allgemeingültig ist, ist es jedoch bei realen Anwendungen meist nötig, in der letzten Verarbeitungsphase den strukturellen Kontext von logischen Objekten bei ihrer Verarbeitung zu beachten [7]. Da sich diese Arbeit und die in ihr entwickelte Designspezifikationsmethodik [em:] gerade mit diesem Arbeitsschritt beschäftigt, werden die strukturellen Kontexte im Folgenden genauer betrachtet.

5.2 Strukturelle Kontexte

5.2.1 Eigenschaften

Ein struktureller Kontext läßt sich über die folgenden drei Merkmale charakterisieren:

- Ein struktureller Kontext wird für ein spezifisches logisches Objekt aufgestellt, für das eine kontextabhängige Verarbeitung erfolgen soll. Dies ist in der Regel dann nötig, wenn für ein logisches Objekt mehrere unterschiedliche Verarbeitungen möglich sind – die Auswahl der geeigneten Verarbeitung kann dann mit Hilfe des strukturellen Kontextes erfolgen.

- Ein struktureller Kontext besteht im Allgemeinen aus einer beliebigen Menge von logischen Objekten eines Dokumentes, die untereinander eine bestimmte Beziehung haben, wie immer sie auch definiert sein mag.

- Ein struktureller Kontext läßt sich für ein gegebenes logisches Objekt zu einem booleschen Wert berechnen; der Kontext ist entweder für ein logisches Objekt vorhanden – oder auch nicht.

Eine Auswahl klassischer Beispiele, die einen strukturellen Kontext zur Auswahl der Verarbeitung eines logischen Objektes erfordern, sind im Folgenden aufgeführt:

1. Formatiere alle Paragraphen mit einem Einzug in der ersten Zeile. Eine Ausnahme hiervon bilden jedoch diejenigen Paragraphen, die unmittelbar einer Überschrift folgen.
2. Die Abschnitte in einem Dokument werden mit arabischen Ziffern durchnummeriert. Im Anhang werden sie jedoch mit großen lateinischen Buchstaben bezeichnet.
3. Hervorgehobener Text wird kursiv gesetzt. Hervorgehobener Text in einer Hervorhebung hingegen wird wieder in normaler Schrift gesetzt.
4. Der erste Eintrag in einer nummerierten Liste wird mit der Ziffer 1 versehen, die folgenden Einträge jeweils mit einer Nummer, die um 1 größer ist als beim vorherigen Eintrag der Liste.

5.2.2 Strukturelle Kontexte in bestehenden Systemen

Word-Processing Systeme

Auch in bestehenden Word-Processing Systemen ist es nötig, Dokumentbestandteile bezüglich ihres strukturellen Kontextes zu verarbeiten. Als Diskussionsgrundlage und zur Verdeutlichung der in diesen Systemen anzutreffenden Situation bei der Verarbeitung von logischen Objekten wird in diesem Abschnitt als konkretes Produkt Microsoft Word verwendet.

Wie bereits in 2.2.2 beschrieben wurde, verfügt auch Microsoft Word in Ansätzen über die Möglichkeit, den Inhalt eines in ihm erstellten Dokumentes bezüglich der logischen Gliederung auszuzeichnen. Insbesondere können in Microsoft Word die einzelnen Absätze, die ein Dokument enthält, verschiedenen Kategorien zugeordnet werden – eine unabdingbare Voraussetzung für eine kontextabhängige Verarbeitung von Dokumentbestandteilen.

Mit den in Microsoft Word verfügbaren Kategorien für die logischen Objekte sind jedoch *unmittelbar* einzelne Designregeln verknüpft, die ohne die Möglichkeit einer weiteren Bedingungsabfrage auf die entsprechenden Absätze zur Verarbeitung angewendet werden. Eine vom strukturellen Kontext eines logischen Objektes abhängige Verarbeitung (besondere Verarbeitung des ersten Absatzes nach einer Überschrift) ist somit in Microsoft Word per se nicht durchführbar.

Aus diesem Grund wird hier ein Weg gewählt, der den verfügbaren Mechanismus der *reinen* logischen Auszeichnung *mißbraucht*, um den strukturellen Kontext von logischen Objekten in einer Verarbeitung zu simulieren: Für die logischen Objekte, die zwar von ihrer logischen Aufgabe her gleich sind, sich aber in verschiedenen Kontexten befinden, werden variante Namen verwendet. Man spricht an dieser Stelle auch von einem *illogical logical markup*.

Dieser Ausweg der *un*logischen Auszeichnung, in der logische Objekte nicht nur bezüglich ihrer eigentlichen logischen Aufgabe, sondern auch bezüglich einer nötigen *Spezialverarbeitung* ausgezeichnet werden, stößt jedoch schnell an seine Grenzen. Die in einem Dokument

enthaltene Auszeichnung ist bei dieser Vorgehensweise exakt an dieser einen Spezialverarbeitung ausgerichtet, weitere Verarbeitungen, die zum Zeitpunkt der Dokumenterstellung noch nicht bedacht waren, sind auf dieser Grundlage mit Microsoft Word nicht ausführbar.

Auf logischer Auszeichnung basierende Systeme

Mit Hilfe der aktuell verfügbaren Editoren für logisch ausgezeichnete Dokumente, die es ermöglichen, Dokumente gemäß den Standards SGML und XML zu erfassen, ist es natürlich möglich, rein logisch ausgezeichnete Dokumente zu erstellen. Aber auch bei diesen Produkten stellt sich die Frage, über welches Verarbeitungsmodell sie verfügen, um den strukturellen Kontext von logischen Objekten bei ihrer Verarbeitung berücksichtigen zu können.

So verfügt beispielsweise der ADEPT Editor von Arbortext[3] zunächst nur über den selben einfachen Zuordnungsmechanismus von Verarbeitungsanweisungen an logische Objekte wie Microsoft Word. Unterschiedliche Verarbeitungen für gleich benannte logische Objekte können hier im Gegensatz zu Microsoft Word jedoch mit Hilfe eines speziellen Mechanimus erreicht werden: Er ermöglicht, dass anstelle der normalerweise verwendeten allgemeinen Designregel für eine Klasse von logischen Objekten für bestimmte von Hand ausgewählte logische Objekte individuelle Designregeln verwendet werden.

Somit ist in diesem Fall zumindest das erstellte Dokument nicht direkt von einer an die Verarbeitung angepassten *un*logischen Auszeichnung betroffen. Trotzdem müssen all diejenigen Dokumentbestandteile speziell von Hand markiert werden, die eine individuelle Verarbeitung erfordern. Ebenso wie in Microsoft Word kann ohne eine Anpassung von Hand ein Dokument aufgrund eines fehlenden Mechanimus, der den strukturellen Kontext von logischen Objekten bei der Verarbeitung beachtet, nicht automatisch bezüglich neuer struktureller Anforderugen verarbeitet werden.

Anders gestaltet sich die Situation in Systemen, die auf den neuen Standards wie CSS1, CSS2 und DSSSL aufbauen. Hier ist es möglich, den strukturellen Kontext eines logischen Objektes bei seiner Verarbeitung als ein Kriterium zur Auswahl der anzuwendenden Designregel zu verwenden: Ein wichtiges Kriterium für die vollkommen automatische Verarbeitung von logisch ausgezeichneten Dokumenten, die ohne ein Zutun von Menschen abläuft. Die in diesen Standards spezifizierbaren strukturellen Kontexte können dabei gewisse Bereiche im gegebenen Dokument umfassen, die immer von dem betroffenen logischen Objekt ausgehen. Die dabei möglichen Formen sind graphisch in Abbildung 5.3 dargestellt.

Die in Abbildung 5.3 unter den Punkten a) und b) dargestellten graphischen Formen der möglichen spezifizierbaren Kontextbedingungen zeigen deutlich auf, dass die beiden CSS-Standards CSS1 und CSS2 für Online-Anwendungen entwickelt wurden, bei denen Dokumente über ein Rechnernetz an die Nutzer übermittelt werden. Hier ist es aufgrund der unvorhersehbaren Übertragungsraten nötig, die Dokumentbestandteile unmittelbar bei ihrer Ankunft in einem Zielsystem für die Nutzer darzustellen: Ein Dokument mit 20 Seiten Umfang, von denen bereits 19 Seiten angekommen sind, kann einen Nutzer bereits für seine nächste Zeit in

[3] URL: http://www.arbortext.com

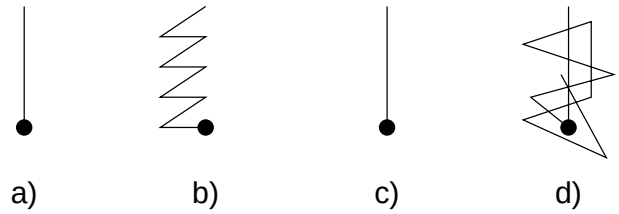

Abbildung 5.3: Formen von Kontextbedingungen: Der schwarze Kreis stellt jeweils das logische Objekt dar, für das der Kontext spezifiziert wird. a) stellt den möglichen Kontextbereich für CSS1 Bedingungen dar. Es sind ausschließlich die Vorgänger in einer Kontextbedingung verwendbar. In b) wird der mögliche Bereich für CSS2 Bedingungen dargestellt. Er umfasst neben den Vorgängern auch deren linke Geschwister. In c) und d) wird der mögliche Bereich für DSSSL Kontextbedingungen aufgezeigt. c) ist dabei identisch zu a) und visualisiert die Möglichkeiten, die in DSSSL mit einfachen Mitteln zu erreichen sind. Die in d) dargestellte Variante ist nur mit größerem Aufwand erzielbar; hier können beliebige logische Objekte den strukturellen Kontext bilden.

Anspruch nehmen und ihm dabei wichtige Informationen zur Verfügung stellen – jedoch nur dann, wenn es für ihn auch verfügbar ist. Aus diesem Grund stützen sich strukturelle Kontextbedingungen in den CSS Standards nur auf beim Eintreffen eines logischen Objektes bereits verfügbare Informationen ab; zukünftige logische Objekte können hier in Kontextbedingungen nicht berücksichtigt werden (siehe hierzu auch Abbildung 5.4). Dies ist eine Eigenschaft die bedingt, dass z.B. für ein letztes logisches Objekt in einer Umgebung (z.B. einer Tabelle) keine besondere Verarbeitung bezüglich seiner Eigenschaft, letztes Objekt zu sein, spezifiziert werden kann.

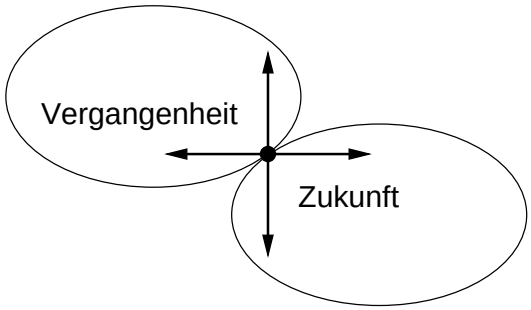

Abbildung 5.4: Erlaubte Bereiche für online-fähige Kontextbedingungen: Eine Kontextbedingung ist nur dann online-fähig, wenn sie ausschließlich Informationen benötigt, die zum frühestmöglichen Verarbeitungszeitpunkt eines betroffenen logischen Objektes schon bekannt sind (Vergangenheit).

5.2.3 Ausdrucksstärke von strukturellen Kontexten

Für jedes konkrete System, das strukturelle Kontexte zur Auswahl der geeigneten Verarbeitungsanweisungen für logische Objekte verwendet, gilt es zu entscheiden, welche Ausdrucks-

stärke die verwendbaren strukturellen Kontextbedingungen haben können. Prinzipiell steht hier natürlich ein Lösungsbereich offen, der von ganz einfachen strukturellen Kontexten bis hin zu komplexen Kontexten reicht.

Wie immer bringen auch hier die extremen Möglichkeiten des Lösungsbereiches aber verschiedene Vor- und Nachteile auf verschiedenen Ebenen mit sich. So werden aus *technischer Sicht* einfache Lösungen schneller und mit weniger Aufwand zu realisieren sein, als die Lösungen für komplexe Kontexte. Aus *Benutzersicht* wird sich analog hierzu die Verwendung und Spezifikation der einfachen Kontexte ebenfalls einfacher als die Verwendung von komplexen Kontexten gestalten.

An dieser Stelle gilt es also vorsichtig abzuwägen, welchen Lösungsweg man für die strukturellen Kontexte wählen soll, die in den Umgebungsprädikaten von [em:] verwendet werden können. Aufgrund der in diesem Arbeitsbereich noch fehlenden Praxisergebnisse der bereits verfügbaren Kontextansätze wird für diese Arbeit ein pragmatischer Weg gewählt, der eine 90%-Lösung darstellen soll: Mit dem für [em:] zu entwerfenden Kontextmodell sollen die *gängigen* Aufgabenstellungen der Dokumentverarbeitung, die eine kontextabhängige Verarbeitung erfordern, lösbar sein; die *ungewöhnlichen* Aufgabenstellungen sollen hier nicht betrachtet werden.

Aufgrund der im Verlauf der Entwicklung des [es:]-Systems gesammelten Erfahrungen wird deshalb im Folgenden ein Kontextmodell vorgestellt, das Kontexte ermöglicht, die ausschließlich logische Objekte beinhalten, die *lokal* zu dem logischen Objekt sind, für das ein Kontext aufgestellt werden soll. Die Eigenschaft lokal zu einem gegebenen logischen Objekt zu sein erfüllen in [es:] all diejenigen logischen Objekte, die Geschwister oder Vorfahren des logischen Ausgangsobjektes sind. Der Begriff der Lokalität eines logischen Objektes zu einem gegebenen logischen Objekt wird in Abbildung 5.5 graphisch veranschaulicht.

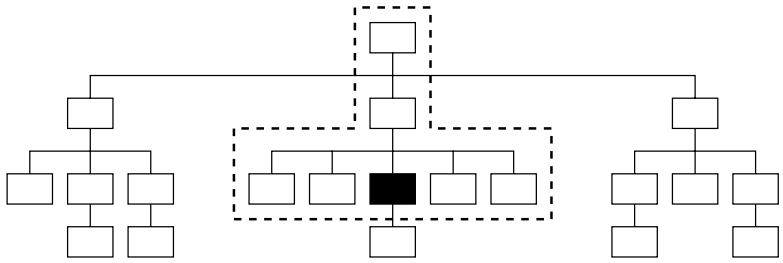

Abbildung 5.5: Lokale Umgebung eines logischen Objektes: In obiger Darstellung ist die lokale Umgebung für das schwarz markierte logische Objekt gestrichelt umrandet eingezeichnet.

Die Abstützung auf Kontexte, die nur lokale logische Objekte für ein zu verarbeitendes logisches Objekt beinhalten dürfen, scheint auch vernünftig zu sein: Viele reale Anwendungsfälle für eine notwendigerweise kontextabhängige Verarbeitung von Dokumentbestandteilen (darunter auch die in 5.2.1 aufgeführten fünf Beispiele) lassen sich mit strukturellen Kontexten beschreiben, die nur lokale logische Objekte beinhalten.

5.3 Das T-Kontextmodell

Das im Laufe dieses Abschnitts zu entwickelnde T-Kontextmodell soll im Rahmen der Designspezifikationsmethodik [em:] zur Spezifikation von Umgebungsprädikaten eingesetzt werden können. Wie bereits weiter oben erwähnt wurde, soll bei seinem Entwurf ein pragmatischer Weg gewählt werden, der eine 90%-Lösung darstellt: Gängige Aufgabenstellungen sollen mit seiner Hilfe gelöst werden können. Aus diesem Grund wird im Folgenden ein auf Abbildung 5.5 aufbauendes Kontextmodell entwickelt, dessen Name sich aus der graphischen Form (ein umgedrehtes T) eines möglichen Kontextes ergibt: Das T-Kontextmodell.

Im Rahmen der Einführung des T-Kontextmodells wird an verschiedenen Stellen auf die *T-Erweiterung* eines logisch ausgezeichneten Dokumentes Bezug genommen, die aus diesem Grund hier nun zentral eingeführt wird.

T-Erweiterung von Dokumenten

Bei der Diskussion um die kontextabhängige Verarbeitung der logischen Objekte eines logisch ausgezeichneten Dokumentes wird bei Verwendung des Begriffes logisch ausgezeichnetes Dokument insbesondere Bezug auf die logische Struktur des Dokumentes genommen. Diese ist in Definition 6 dieser Arbeit festgelegt worden. Die T-Erweiterung eines logisch ausgezeichneten Dokumentes erweitert nun diese logische Struktur eines gegebenen logisch ausgezeichneten Dokumentes um die sogenannten *logischen Endemarker*, von denen drei verschiedene Klassen existieren. Die einzige Instanz eines *Top*-Endemarkers wird oberhalb des logischen Wurzelelementes angeordnet. Die eingebrachten Instanzen der *Links*-Endemarker werden jeweils links von dem am weitesten links liegenden Geschwister einer Gruppe von logischen Objekten eingebracht. Analoges gilt für die *Rechts*-Endemarker.

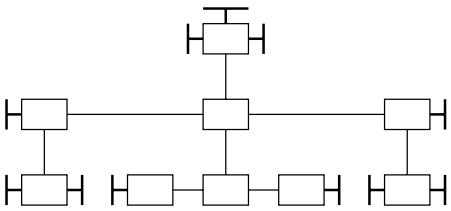

Abbildung 5.6: T-Erweiterung eines logisch ausgezeichneten Dokumentes: Zur Verdeutlichung der Geschister-Beziehung sind im Gegensatz zu Abbildung 2.3 in dieser Abbildung Geschwister direkt mit horizontalen Linien verbunden.

5.3.1 T-Umgebung

Im T-Kontextmodell ist eine T-Umgebung $\mathcal{U}_{ot}^{S}$ derjenige Subgraph (U_{ot}, S) eines T-erweiterten logisch ausgezeichneten Dokumentes D, der für ein logisches Objekt o zur Überprüfung auf das Vorhandensein eines strukturellen T-Kontextes verwendet werden kann.

Dieser Subgraph umfaßt zunächst das *logische Basisobjekt* o, für das ein zu überprüfender T-Kontext aufgestellt ist. Neben diesem logischen Basisobjekt o beinhaltet eine T-Umgebung weiterhin auch all die logischen Objekte w, die ein *Vorfahre* oder ein *Geschwister* des logischen Basisobjektes o im gegebenen logisch ausgezeichneten Dokument D sind. Mit Hilfe der bisher getroffenen Festlegung ergibt sich somit für jedes logische Objekt o in einem logisch ausgezeichneten Dokument D die Menge

$$U_o := \{w \in D : w \text{ ist } o \text{ oder } w \text{ ist ein Vorfahre von } o \text{ oder } w \text{ ist ein Geschwister von } o\}$$

von logischen Objekten (siehe hierzu auch Abbildung 5.7 a), die zu einer T-Umgebung beitragen.

Aus Gründen einer größeren Ausdrucksstärke für die noch einzuführenden T-Kontexte werden neben den bisher durch U_o zugelassenen logischen Objekten auch die *logischen Endemarker* für die Geschwister bzw. Vorgänger des logischen Basisobjektes aus dem T-erweiterten Dokument mit in die T-Umgebung aufgenommen. Sie markieren jeweils explizit das linke, rechte und obere Ende einer T-Umgebung (siehe hierzu auch Abbildung 5.7 b) und bilden die Menge

$$U_t := \{\vdash, \dashv, \top\}.$$

Insgesamt besteht somit die Menge der in einer T-Umgebung $\mathcal{U}_{ot}^S$ enthaltenen Objekte aus der Vereinigung der beiden Mengen U_o (den zugelassenen logischen Objekten aus D) und U_t (den logischen Endemarkern):

$$U_{ot} := U_o \cup U_t.$$

In einer T-Umgebung $\mathcal{U}_{ot}^S$ ist nun nicht nur die Menge der beteiligten Objekte von Bedeutung, sondern auch ihre strukturelle Anordnung S zueinander: Eine T-Umgebung ist ein Graph. Aus diesem Grund umfaßt eine T-Umgebung $\mathcal{U}_{ot}^S$ neben der Menge U_{ot} auch noch die Information über die strukturelle Anordnung der logischen Objekte und Endemarker in dem T-erweiterten logisch ausgezeichneten Dokument. Sie ergibt sich über die Menge der Kanten S, die die logischen Objekte und die Endemarker in T-erweiterten Dokumenten miteinander verbinden. Eine T-Umgebung $\mathcal{U}_{ot}^S$ ist somit über das Tupel (U_{ot}, S) definiert.

5.3.2 T-Kontexte

Nachdem nun in 5.3.1 formal die T-Umgebung $\mathcal{U}_{ot}^S$ eines logischen Objektes o als derjenige Subgraph in einem T-erweiterten Dokument festgelegt wurde, in dem die strukturelle Kontextbedingung für o vorkommen kann, gilt es nun in diesem Abschnitt formal die T-Kontexte $\mathcal{K}_b$ einzuführen, die diese strukturelle Kontextbedingung spezifizieren und zur Überprüfung mit Hilfe eines T-Matching-Algorithmus verwendet werden können.

Ein T-Kontext $\mathcal{K}_b$ spezifiziert eine strukturelle Kontextbedingung in Form eines Graphen G, der die Gestalt eines auf dem Kopf stehenden T hat. Formal betrachtet ist ein T-Kontext $\mathcal{K}_b$ ein Graph G, der über das Viertupel (V, E, L, b) beschrieben wird. V ist dabei die Menge der im Graph G enthaltenen Knoten, $E \subseteq V \times V$ die Menge von Kanten zwischen den Knoten in V, L eine Beschriftungsfunktion für die Kanten und Knoten und b ein ausgezeichneter Basisknoten.

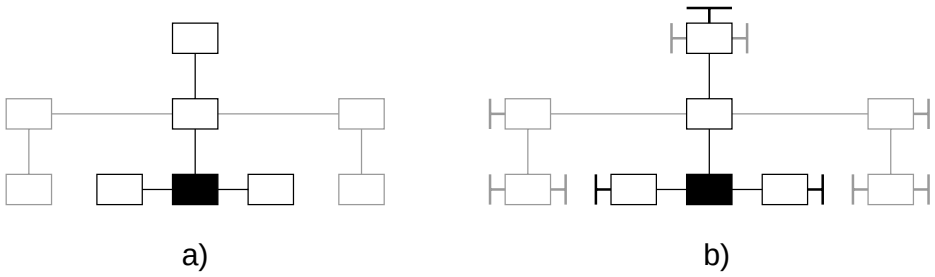

Abbildung 5.7: Beispiel einer T-Umgebung für ein logisches Objekt (schwarz ausgefüllt) in einem Dokument: In Grau sind logische Objekt dargestellt, die im Dokument enthalten sein können, aber nichts zur T-Umgebung beitragen. Zur Verdeutlichung, dass die horizontal angeordneten logischen Objekte Geschwister sind, werden sie anstelle der üblichen Darstellung (siehe Abbildung 2.3) untereinander mit horizontalen Linien verbunden; der Eltern-Kind Aspekt ist nur für das logische Basisobjekt (schwarz dargestellt) von Bedeutung. Die in b) dargestellten Endemarker des T-erweiterten Dokumentes aus a) zeigen *explizit* das nicht weitere Vorhandensein von logischen Objekten in die jeweilige Richtung an.

Die Menge V der Knoten beinhaltet neben einer beliebigen Anzahl von allgemeinen Knoten als Basis grunsätzlich die vier ausgezeichneten Knoten b, $\vdash$, $\dashv$ und $\top$, die den *Basisknoten*, *Links-Endeknoten*, *Rechts-Endeknoten* und *Top-Endeknoten* darstellen. Von dem Knoten b gehen exakt drei Kanten auf drei verschiedene Knoten aus, von den drei Endeknoten keine und von allen anderen Knoten jeweils exakt eine gerichtete Kante. Die Beschriftungsfunktion L weist einerseits mit Ausnahme der Endeknoten $\vdash$, $\dashv$ und $\top$ allen Knoten aus V einen String zu. Andererseits weist sie allen Kanten aus E eine 1 oder ein $+$ zu.

G besteht somit aus drei gerichteten Subgraphen die *Stämme* genannt werden und zusammen eine Überdeckung von G ergeben. Die drei Stämme haben genau den Knoten b gemeinsam; ansonsten sind sie gegenseitig disjunkt. Der *Top*-Stamm ist ein Digraph (V_U, E_U), wobei $V_U \subset V$ ist und weiterhin explizit gilt, dass $b, \top \in V_U$ sowie $\vdash, \dashv \notin V_U$. E_U legt aufgrund der Eigenschaft, dass von jedem allgemeinen Knoten exakt eine gerichtete Kante ausgeht, eine Linie von b nach $\top$ fest. Der *Links*-Stamm (V_L, E_L) und der *Rechts*-Stamm (V_R, E_R) sind analog zum *Top*-Stamm definiert und haben somit zu ihm vergleichbare Eigenschaften.

Anschaulich kann man sich einen T-Kontext wie in Abbildung 5.8 dargestellt vorstellen. Die strukturelle T-Form des dort dargestellten Graphen ist zwar willkürlich gewählt, sie genügt aber inhaltlich der oben getroffenen Nomenklatur für die Endemarker und entspricht der Form, in der sie im T-Matching-Algorithmus verwendet werden wird.

Offen ist an dieser Stelle noch die Semantik, die ein T-Kontext $\mathcal{K}_b$ trägt. Sie ergibt sich erst über den T-Matching-Algorithmus, der im nächsten Abschnitt eingeführt wird.

5.3.3 T-Matching-Algorithmus

Die Semantik eines T-Kontextes $\mathcal{K}_b$ ist bisher nicht festgelegt worden. Sie ergibt sich jedoch im Folgenden durch die informelle Vorstellung eines T-Matching-Algorithmus, der einen T-

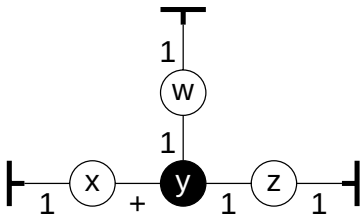

Abbildung 5.8: Beispiel eines T-Kontextes: Zentral im Graphen ist der Basisknoten b (schwarz gefüllt) angebracht. Von ihm gehen T-förmig die drei Stämme ab, deren Knoten über Kanten miteinander verbunden sind und die jeweils mit einem Endeknoten enden. Die Beschriftung der Knoten steht in den Knoten, die Beschriftung der Kanten neben oder unter ihnen.

Kontext $\mathcal{K}_b$ nimmt und prüft, ob er in der T-Umgebung $\mathcal{U}_{ot}^S$ für ein gegebenes logisches Objekt o gegeben ist.

T-Einfügungen von T-Kontexten

Genauer betrachtet wird mit Hilfe eines T-Matching-Algorithmus überprüft, ob ein gegebener T-Kontext K_b in eine T-Umgebung $\mathcal{U}_{ot}^S$ für ein logisches Objekt o *T-eingefügt* werden kann. Die T-Einfügung wird dabei durch eine Abbildung der Knoten des gegebenen T-Kontextes auf die logischen Objekte der T-Umgebung durchgeführt, wobei die Kanten des T-Kontextes $\mathcal{K}_b$ mit der Beschriftung $+$ *gestreckt* werden können. Exakt sind folgende Schritte nötig, um alle passenden T-Kontexte K_b einer gegebenen T-Kontextmenge für ein logisches Objekt o mit einer Umgebung $\mathcal{U}_{ot}^S$ zu finden, die T-eingefügt werden können:

Sortiere alle T-Kontexte aus, für die nicht gilt: Beschriftung(b) = Name(o).
Für die restlichen T-Kontexte gehe wie folgt vor:

- Bilde den Basisknoten b auf das Basisobjekt o ab.
- Bilde die drei Endknoten auf die entsprechenden Endmarker ab.
- Bilde unter Beachtung der Kantenbeschriftung die restlichen Knoten des gegebenen T-Kontextes auf die logischen Objekte ab. Eine Kantenbeschriftung von 1 des T-Kontextes bedeutet, dass die Kante genau auf *eine* Kante der T-Umgebung passt, eine Kantenbeschriftung von $+$, dass die Kante auf *mehrere* Kanten der T-Umgebung gestreckt werden kann.

Ein T-Kontext K_b ist für ein logisches Objekt o erfüllt, wenn für ihn eine T-Einfügung gemäß obigem Vorgehen gefunden werden kann. Mit Hilfe des Konstrukts der T-Einfügung ergibt sich somit auch unmittelbar auf informativer Ebene die Semantik von T-Kontexten. Abbildung 5.9 zeigt einige Beispiele für T-Kontexte auf.

Nach dieser informellen Einführung der Semantik von T-Kontexten ist erklärbar, aus welchem Grund in einer T-Umgebung $\mathcal{U}_{ot}^S$ die logischen Endemarker aufgenommen werden: Ohne

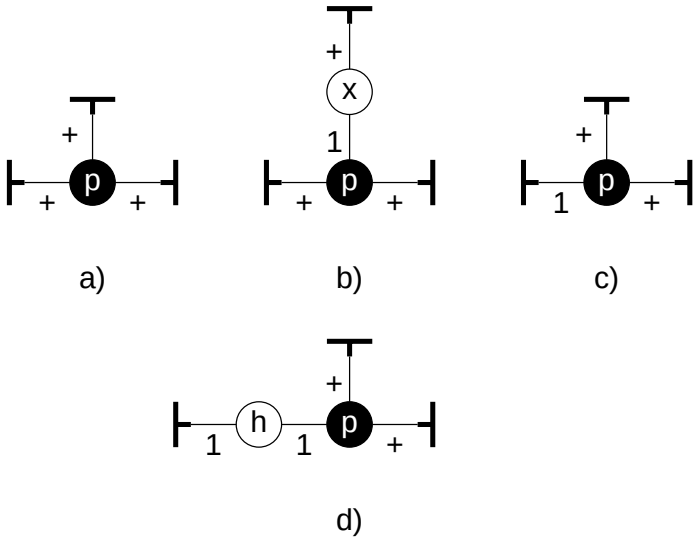

Abbildung 5.9: Konkrete Beispiele für T-Kontexte: a) Ein T-Kontext, der für jedes logische p Objekt erfüllt ist. b) Ein T-Kontext, der für die logischen p Objekte passt, die unmittelbar in einem logischen x Objekt vorkommen. c) Ein T-Kontext, der für ein logisches p Objekt zutrifft, der kein logisches Objekt als linkes Geschwister hat. d) Ein T-Kontext, der für das erste logische p Objekt nach einem logischen h Objekt passt.

ihr Vorkommen in der T-Umgebung eines logischen Objektes o könnte für dieses logische Objekt nicht explizit getestet werden, ob es ein erstes oder letztes Objekt unter seinen Geschwistern ist.

Komplexität von T-Kontexten

Ein Problem, das sich bei dem Verfahren der T-Einfügung nach obigem Vorgehen ergibt, ist die Tatsache, dass für ein logisches Objekt o gleichzeitig mehrere T-Kontexte erfüllt sein können. Im Rahmen des Einsatzes der T-Kontexte, der Auswahl einer geeigneten Designregel zur Verarbeitung eines logischen Objektes, ist jedoch möglichst eine *Eindeutigkeit* bei der Auswahl eines T-Kontextes verlangt: Ein T-Kontext soll aufgrund sinnvoller Kriterien *der* eindeutige Gewinner sein.

Als sinnvoller Ansatz zur Lösung dieser Problemstellung hat sich herausgestellt, immer den *speziellsten* T-Kontext aller möglichen T-Kontexte für ein logisches Objekt zu wählen. So ist in der Realität ein T-Kontext, der den Kontext „erster logischer Paragraph nach einer Überschrift“ (siehe Abbildung 5.9 d) beschreibt, spezieller als ein allgemeiner *Standard*-Kontext für einen Paragraph (siehe Abbildung 5.9 a). Um festzustellen, wie speziell ein gegebener T-Kontext ist, wird ein Maß definiert, die *Komplexität*. Falls für ein logisches Objekt mehrere T-Kontexte passen, wird derjenige ausgewählt, der die maximale Komplexität hat. Gibt es mehrere solcher Regeln mit maximaler Komplexität (ein Fall, der als Designfehler eines Nutzers bewertet werden kann), wird aus diesen ein beliebiger ausgewählt.

Die Komplexität eines T-Kontextes ist eine natürliche Zahl, die nach folgender Heuristik berechnet wird:

- Jeder zu einem T-Kontext gehörige Knoten (einschließlich Basisknoten) trägt 1 zur Komplexität bei (je mehr Knoten ein T-Kontext besitzt, desto spezieller ist er).
- Jede mit 1 beschriftete Kante erhöht die Komplexität um 1 (da explizit die unmittelbare Nachbarschaft von zwei logischen Objekten bzw. einem logischen Objekt und einem Endeobjekt verlangt wird).

5.4 Spezifikation von strukturellen Kontexten

5.4.1 Grundsätzliche Spezifikationsmöglichkeiten

In der Designspezifikationsmethodik [em:] ist die Anwendung von Designregeln an die Erfüllung von Umgebungsprädikaten geknüpft. Somit ist es nötig, die Nutzer des Systems in die Lage zu versetzen, Umgebungsprädikate spezifizieren zu können. Hierzu sind unabhängig von der Art des Kontextmodells zwei grundlegende Ansätze möglich: Zum einen kann den Nutzern eine Liste von möglichen Umgebungsprädikaten zur Auswahl eines geeigneten Prädikates angeboten werden. Zum anderen kann man ihnen Möglichkeiten zum freien Aufstellen von Kontexten anbieten.

Bei der ersten Variante, der Realisierung einer Auswahlliste für Umgebungsprädikate, ist es bei einer Systemerstellung nötig, die zur Verfügung zu stellenden Umgebungsprädikate *vorzuanalysieren*. Dies ist zwar insbesondere für Standardaufgaben möglich und sinnvoll, insgesamt wird dieser Ansatz aber einschränkend sein, da er nicht alle in einem Kontextmodell möglichen Kontexte zur Verfügung stellen kann – das Kontextmodell wird nicht ausgeschöpft.

Bei der zweiten Variante, dem freien Aufstellen von Kontexten durch die Nutzer, wird hingegen von den Nutzern verlangt, dass sie die benötigten Kontexte selbst konstruktiv angeben. Durch dieses Vorgehen sind auch Kontexte spezifizierbar, die selten vorkommen und nicht vorhersehbar sind – das zur Verfügung gestellte Kontextmodell kann von den Nutzern voll ausgeschöpft werden.

Das zur freien Aufstellung von strukturellen Kontexten übliche Verfahren ist die textuelle Aufschreibung der Kontexte in einer speziellen programmiersprachlichen Notation (siehe z.B. [53][54][30]). Dies entspricht jedoch nicht dem Ansatz, der in dieser Arbeit verfolgt werden soll. Hier soll vielmehr eine Möglichkeit geschaffen werden, die die Nutzer des Systems von jeglicher textueller Kontextspezifikation abschottet: Umgebungsprädikate sollen hier durch graphische Interaktion spezifiziert werden können.

5.4.2 Interaktiv graphische Spezifikation

Die Aufgabenstellung bei der interaktiv graphischen Spezifikation eines strukturellen Kontextes ist einfach zu beschreiben: Ein Nutzer eines Systems will ein strukturelles Kontextprädikat für

eine gewisse strukturelle Bedingung (die gegebene Aufgabe) aufstellen. Diese Bedingung muss nun zunächst vom Nutzer verstanden, modelliert und verinnerlicht werden: Der Nutzer baut sich ein mentales Modell für die von ihm verstandene Aufgabenstellung auf. Ausgehend von diesem Modell des von ihm verstandenen Aufbaus der Aufgabenstellung kann der Nutzer mit Hilfe einer gegebenen Software den von ihm zu modellierenden Kontext erstellen (siehe auch Abbildung 5.10).

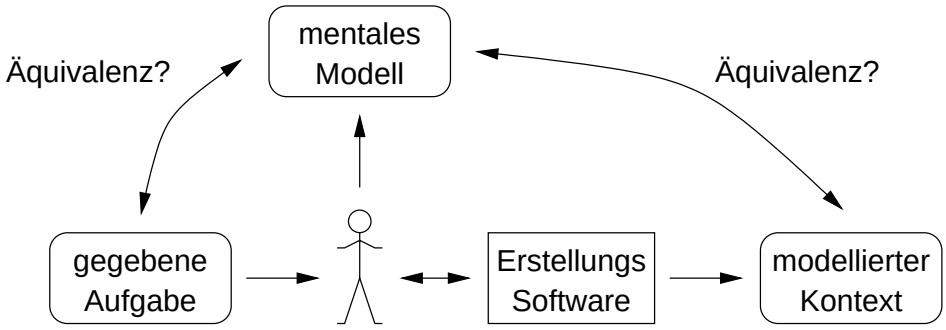

Abbildung 5.10: Prozess der Kontextmodellierung

Bei diesem Prozess der interaktiv graphischen Spezifikation von Kontexten tritt an zwei Stellen die Frage nach der Äquivalenz der gegebenen Aufgabenstellung und ihrer Modellierung auf. Dies ist zunächst die Aufnahme der gegebenen Aufgabenstellung durch den Nutzer – ein Punkt, der aufgrund seiner psychologischen Natur in dieser Arbeit nicht weiter betrachtet werden soll. Zum anderen ist dies der Übergang des mentalen Modells des Nutzers in den modellierten Kontext.

Bei diesem zweiten Übergang tritt für einen Nutzer das Problem auf, dass er mit Hilfe der über die zur Verfügung gestellte Software und der ihr zugrundeliegenden Spezifikationsmethodik einen strukturellen Kontext erstellen muss, der seinen Vorstellungen entspricht. Er muss sich somit zum einen der Semantik seiner möglichen Interaktionen mit dem System bewusst sein sowie zum anderen die Semantik der von dem System aufgrund seiner Interaktion gelieferten graphischen Anzeige kennen. Aus diesem Grund muss sowohl die *Semantik der möglichen Interaktionen* als auch die *Semantik der Anzeige* eines erstellten Kontextes eindeutig für ein gegebenes System definiert sein.

5.4.3 Interaktiv graphische Spezifikation von T-Kontexten

Strukturelle T-Kontexte werden interaktiv graphisch anhand eines gegebenen logisch ausgezeichneten Dokumentes (siehe z.B. Abbildung 5.5 auf Seite 165) erstellt. An dieser Stelle der Arbeit wird nun das grundlegende Vorgehen hierzu vorgestellt, die exakte Umsetzung in einer graphischen Benutzungsoberfläche wird in Abschnitt 6.2.11 eingeführt.

T-Kontexte bestehen aus einem beschrifteten Basisknoten, weiteren beschrifteten Knoten und Kanten sowie drei Endeknoten. Es wird somit eine Spezifikationsmethodik mit Interaktionsmöglichkeiten benötigt, mit deren Hilfe exakt obige T-Kontext Bestandteile aufgrund der

Interaktion eines Nutzers mit einem logisch ausgezeichneten Dokument erzeugt werden können.

Spezifikationsmethodik

T-Kontexte werden gemäß dem Paradigma *Design by Example* (siehe auch 4.1.2 und 6.2.5) aufgestellt. Dies bedeutet, dass zur Kontextspezifikation Beispieldokumente verwendet werden, in denen diejenigen logischen Objekte, für die eine spezielle kontextabhängige Verarbeitung erfolgen muss, auch exakt in den benötigten strukturellen Anordnungen vorkommen.

T-Kontexte können dann auf benutzerfreundliche Weise spezifiziert werden, indem durch Interaktion *lediglich* die logischen Objekte im Beispieldokument angewählt werden, die der zu erstellende T-Kontext in Form von mit den Namen der logischen Objekte beschrifteten Knoten (Basisknoten und weitere Knoten) enthalten soll. Da ein T-Kontext mit einem T-Matching-Algorithmus nur auf die T-Umgebung eines logischen Objektes o abgebildet werden kann (siehe 5.3.3) ist offensichtlich, dass dabei nur logische Objekte in der T-Umgebung des logischen Objektes o, für das ein T-Kontext aufgestellt wird, selektiert werden und zum T-Kontext beitragen können.

Aufgrund dieser indirekten Spezifikation der beteiligten Knoten durch ein Anklicken von logischen Objekten durch einen Nutzer können dann bei geeigneten Vorgaben in der Spezifikationsmethodik basierend auf dem gegebenen Beispieldokument die restlichen Bestandteile des T-Kontextes *automatisch* berechnet werden: Die beschrifteten Kanten. Wie in Abschnitt 5.3.2 vorgegeben wird, sind die drei Endeknoten eines T-Kontextes immer ein fixer Bestandteil und müssen folglich nicht berechnet werden.

Leider kann aus dieser Menge der ausgewählten logischen Objekte und der fix vorgegebenen Endeknoten noch kein eindeutiger T-Kontext konstruiert werden (Abbildung 5.11 zeigt ein Beispiel hierzu auf). Es ist nicht klar, in welcher Form die Kanten des zu bildenden Kontextes, die die vom Nutzer durch das Auswählen der logischen Objekte für den Kontext implizierten Knoten verbinden, beschriftet werden sollen.

Um die Spezifikation eines Nutzers, die ausschließlich durch die Auswahl von logischen Objekten erfolgt, eindeutig zu machen, werden bei der Berechnung der Beschriftung der Kanten einfache Regeln einer Heuristik verwendet, die das Paradigma *Design by Example* (jedes logische Objekt, das eine spezielle Verarbeitung benötigt, kommt exakt in der T-Umgebung vor, in der es besonders verarbeitet werden soll) ausnutzt:

- Alle Kanten, die aufgrund zweier unmittelbar benachbarter und selektierter logischer Objekte zwei Knoten im T-Kontext verbinden werden mit einer 1 beschriftet, alle anderen Kanten zwischen zwei Knoten mit einem $+$.
- Die Kanten von einem Knoten zu einem Endeknoten werden dann mit einer 1 beschriftet, wenn der im T-Kontext vorkommende Knoten ein Randknoten ist (das angeklickte logische Objekt ist im T-erweiterten Dokument neben einem Endeobjekt).

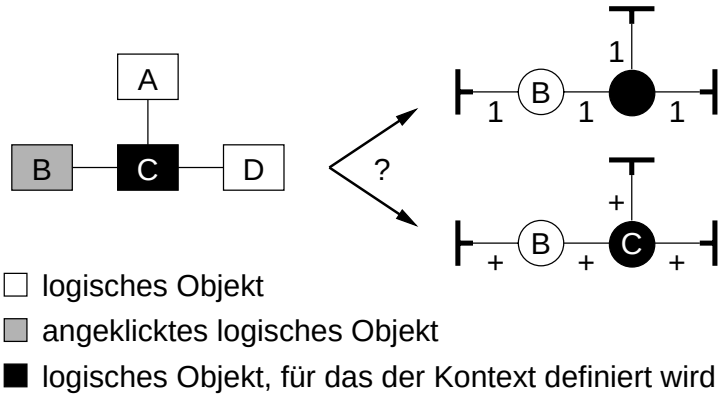

Abbildung 5.11: Beispiel einer graphischen T-Kontext Spezifikation: Obige Abbildung zeigt einen sehr einfachen logischen Baum, in dem eine Auswahl von Knoten selektiert wurde. Aus diesen vorgegebenen Knoten könnten unter anderem die oben dargestellten zwei T-Kontexte berechnet werden.

Die Idee, die hinter den oben aufgestellten zwei Regeln steht, ist, dass *benachbarte selektierte logische Objekte* und *Randpositionen* von logischen Objekten in einem Beispieldokument gezielt zur Erstellung eines T-Kontextes verwendet werden sollen. Für ein logisches Objekt o, das eine Randposition inne hat, kann also auch nur ein T-Kontext spezifiziert werden, der diese Randposition für das logische Objekt o abprüft. Soll ein T-Kontext für das logische Objekt o erstellt werden, der diese Überprüfung nicht beinhaltet, muss o im Beispieldokument auch in einer Nicht-Randposition vorkommen und entsprechend dort für eine Kontextspezifikation verwendet werden. Analoges gilt auch für zwei unmittelbar benachbarte logische Objekte.

Vorteile

Mit dieser die obigen zwei Regeln umfassenden Heuristik wird den Nutzern die Spezifikation von Kontexten durch die automatische Berechnung der Kantenbeschriftungen erleichtert: Die Nutzer müssen lediglich die den Kontext bestimmenden logischen Objekte selektieren. Voraussetzung zur Aufstellung von T-Kontexten für bestimmte logische Objekte ist jedoch immer das Vorliegen eines geeigneten logisch ausgezeichneten Beispieldokumentes. Abschließend wird noch einmal Bezug auf das Beispiel aus Abbildung 5.11 genommen. Gemäß obiger Heuristik würde bei Eingabe der in Abbildung 5.11 dargestellten Objektselektion der in Abbildung 5.12 gezeigte T-Kontext berechnet werden.

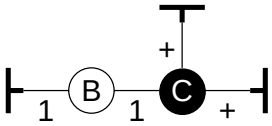

Abbildung 5.12: Berechneter T-Kontext zu Abbildung 5.11: Setzt man `B` mit einem logischen Objekt `Überschrift` und `C` mit einem logischen Objekt `Paragraph` gleich, so spezifiziert der T-Kontext den strukturellen Kontext „erster Paragraph nach einer Überschrift, die kein linkes Geschwister hat“.

Kapitel 6

Die Werkzeuge [pe:] und [de:]

In Kapitel 4 wurde, wie in der Aufgabenstellung dieser Arbeit gefordert, eine Designspezifikationsmethodik entwickelt, die modernen Anforderungen zur Verarbeitung von logisch ausgezeichneten Dokumenten entspricht. In diesem Kapitel wird nun zunächst das Werkzeug [pe:] vorgestellt, das eine funktional vollständige Umsetzung der entwickelten Designspezifikationsmethodik [em:] darstellt. [pe:] ist somit in der Lage, ein logisch ausgezeichnetes Dokument aufgrund einer gegebenen [em:] Designspezifikation in eine Designstruktur zu überführen.

Wäre lediglich das Werkzeug [pe:] gegeben, so müssten [em:] Designspezifikationen von den Nutzern des [es:] Systems von Hand erstellt werden; ein Ansatz, der ausdrücklich in der Aufgabenstellung zu dieser Arbeit augeschlossen wurde – er würde die Nutzer des [es:] Systems in keinerlei Weise geeignet unterstützen. Aus diesem Grund wird in diesem Kapitel weiterhin der im Rahmen dieser Arbeit entwickelte interaktive Designeditor [de:] vorgestellt. Er bildet für seine Nutzer die Konzepte und Objekte von [em:] nach und bietet sie in Form einer graphischen Benutzungsoberfläche zur Manipulation von Designspezifikationen dar – eine Eigenschaft, die derzeit kein anderes allgemein bekanntes System (siehe hierzu z.B. [45]) vorweisen kann.

Diese Benutzungsoberfläche von [de:] soll zunächst einen möglichst intuitiven Einsatz der Designspezifikationsmethodik [em:] ermöglichen, ohne sich vorher in sie mit großem Aufwand einarbeiten zu müssen. Weiterhin soll sie die im Rahmen einer Designspezifikation entwickelten Regeln nicht nur als *leblose*, abstrakte Einheiten zur Bearbeitung anbieten, sondern sie auch im Zusammenhang mit anderen in der Designspezifikation vorkommenden Designregeln im Rahmen einer realen Designapplikation *lebend* und interagierend darstellen; ein Ansatz, der es den Designern einerseits ermöglicht wechselseitige Abhängigkeiten und Auswirkungen von Designentscheidungen besser abzuschätzen und zu verstehen und somit insgesamt zu einem verbesserten Überblick bei der Erstellung von komplexen Designspezifikationen verhilft. Andererseits wird dieser Ansatz Designern auch bei der Einarbeitung und der Übernahme von bereits bestehenden Designspezifikationen helfen.

6.1 Der Designprozessor [pe:]

6.1.1 Überblick

[pe:] ist ein Software-Werkzeug, mit dessen Hilfe die Designspezifikationsmethodik [em:] vollständig umgesetzt wird. Mit seiner Hilfe können gegebene Designspezifikationen auf logisch ausgezeichnete Dokumente angewendet werden, um diese in eine Designstruktur zu überführen. [pe:] bietet dazu zwei Modi an, die unterschiedliche Anforderungen mit sich bringen: einen *Batch*- und einen *Interaktiv*-Modus.

Batch-Modus

Der Batch-Betrieb der [pe:] Komponente des [es:] Systems ist der übliche Einsatz einer zu [pe:] vergleichbaren Komponente in bestehenden Dokumentverarbeitungssystemen. Ein gewöhnliches Ziel einer derartigen Komponente ist die Optimierung auf Geschwindigkeit und auf Speicherplatz. Typisches Anwendungsgebiet für [pe:] ist der Einsatz als Backend in einer Server-Applikation (z.B. einem Print-on-Demand-Server), um Dokumente z.B. aufgrund von Informationen über die zukünftigen Nutzer geeignet aufzubereiten (User Customized Publishing). Abbildung 6.1 stellt dieses Szenario graphisch dar.

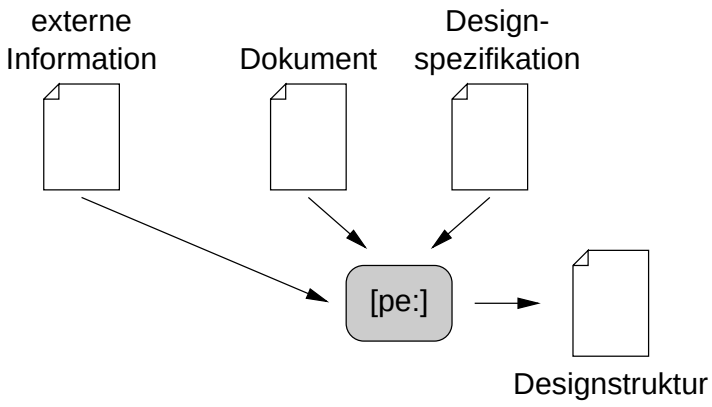

Abbildung 6.1: Der Designprozessor [pe:] in einem Batch-Betrieb Szenario: Basierend auf einem gegebenen Dokument erzeugt der Designprozessor [pe:], gesteuert durch eine Designspezifikation und unter Rückgriff auf externe Information, eine Designstruktur.

Interaktiv-Modus

Im Interaktiv-Modus berechnet [pe:] ebenfalls wie im Batch-Betrieb die Ergebnisse, die im Rahmen einer Designapplikation anfallen. Diese Ergebnise werden jedoch von der [de:] Komponente für die Nutzer des [es:] Systems visualisiert und zu einer benutzerfreundlichen interaktiven Erstellungsmöglichkeit für Designspezifikationen verwendet. [pe:] wird in diesem Modus nicht wie im Batch-Modus lediglich einmal für eine Designapplikation gestartet, um danach

terminieren zu können, sondern wird nach partiellen Änderungen an der aktuell bearbeiteten Designspezifikation immer wieder unmittelbar neue Designapplikationen auf dem gegebenen logisch ausgezeichneten Dokument ausführen – mit dem Ziel, den Designern einen Einblick zu vermitteln, was ihre Änderungen bewirkt haben. Dieser interaktive Modus wird deshalb wie schon der Batch-Modus auf Geschwindigkeit optimiert sein. Im Gegensatz zum Batch-Modus wird im interaktiven Modus die Optimierung bezüglich des Speicherplatzes aber weniger im Vordergrund stehen: Aufgrund schneller Reaktionszeiten des Systems für die Nutzer ist es, wie sich zeigen wird, nötig, im Rahmen einer Designapplikation anfallende Zwischenergebnisse zu speichern. Mit ihrer Hilfe wird es der [de:] Komponente ermöglicht werden, Auswirkungen und Querbeziehungen von Designentscheidungen im Rahmen einer Designspezifikation für die Nutzer derart zu visualisieren, dass sie komplexe Verwendungen von Spezifikationsobjekten und Spezifikationsvariablen- bzw. Spezifikationsattribut-Wertsetzungen vornehmen, analysieren und nachvollziehen können. Abbildung 6.2 zeigt diese Zusammenarbeit zwischen den Komponenten [pe:] und [de:] auf.

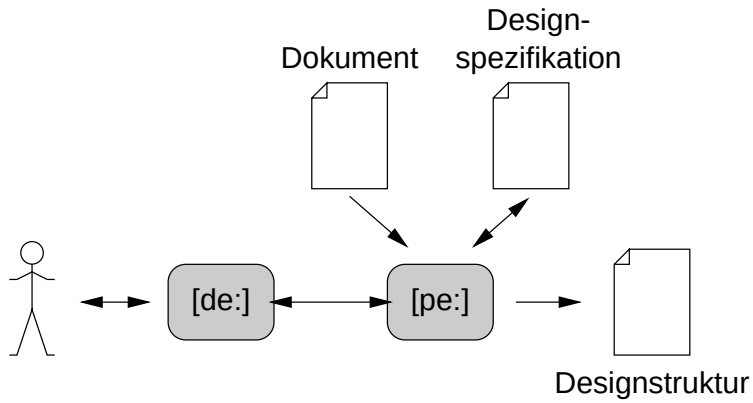

Abbildung 6.2: Zusammenarbeit zwischen [pe:] und [de:]: Im Rahmen einer interaktiven Designspezifikation eines Nutzers mit dem [es:] System dient [de:] ausschließlich als Interaktionskomponente, die sämtliche Berechnungen, die mit einer Designapplikation in Zusammenhang stehen, von [pe:] berechnen läßt. Die Designspezifikation kann hier von [pe:] auch geschrieben werden.

6.1.2 Funktionsumfang von [pe:]

[pe:] unterstützt in vollem Umfang die Designspezifikationsmethodik [em:], wie sie in Kapitel 4 beschrieben worden ist. Somit kann [pe:] die im Folgenden aufgeführten Aufgaben übernehmen und ausführen, die für den Batch-Einsatz des [es:] Systems nötig sind:

- Einlesen eines logischen Dokumentes und Aufbau des entsprechenden logischen Baumes
- Einlesen und verwalten von [em:] Designregeln
- Einlesen und verwalten von Compoundobjekten

- Ausführen des Matching-Algorithmus
- Anwenden und Ausführen von Designregeln
- Aufbau einer Designstruktur

Bei der Ausführung der genannten Aufgaben wird teilweise Rückgriff auf einen in [pe:] realisierten Interpreter genommen, der die in 4.9 eingeführte [em:]-Ausdruckssprache versteht und gleichzeitig die Ausführung der Berechnung von [em]-Ausdrücken durchführen kann. Dieser Interpreter stützt sich dabei auf eine Symboltabelle ab, die den in Abschnitt 4.4.5 vorgestellten Spezifikationsvariablenfluss sowie den in Abschnitt 4.4.6 vorgestellten logischen Fluss in erweiterten Designstrukturen umsetzt. Als Konsequenz daraus entsprechen die Berechnungen des Interpreters exakt dem Modell, wie es in [em:] durch Verschattungskonzepte und die Wirkung von Spezifikationsobjekten vorgegeben wird.

Im Interaktiv-Modus bietet [pe:] mit Hilfe geeigneter Datenstrukturen weiterhin die Möglichkeit an, das Ergebnis von Abhängigkeitsbeziehungen zwischen Spezifikationsvariablen- und Spezifikationsattribut-Werten, die im Rahmen einer Designapplikation aufgrund des Spezifikationsvariablenflusses und des logischen Flusses entstehen, zu berechnen und festzuhalten.

6.1.3 Funktionsweise von [pe:]

Die [pe:] Komponente des [es:] Systems kann bezüglich einem von zwei Modi aktiv sein. Im Folgenden wird zunächst ihre Funktionsweise im Batch-Betrieb näher untersucht.

Batch-Betrieb

Einlesen eines Dokumentes:
Zu Beginn des Ablaufs einer durch die [pe:] Komponente ausgeführten Designapplikation wird das der Komponente vorgegebene Dokument eingelesen. Dies geschieht über eine neutrale Schnittstelle, über die ein Dokument, das gemäß der ESIS-Form[1] ausgezeichnet ist, eingelesen wird.

Dieses Vorgehen öffnet dem System [es:] zum einen auf elegante Weise den einfachen Zugriff auf beliebige gemäß dem XML und dem SGML Standard ausgezeichnete Dokumente: Der frei verfügbare und nach allgmeiner Meinung in Bezug auf die Standards XML und SGML vollständigste Parser NSGMLS [14] liefert das ESIS-Format als eine mögliche Ausgabe. Zum anderen bietet dieses Vorgehen der Verwendung einer neutralen Einleseschnittstelle die Möglichkeit, auch Dokumente, die gemäß weiterer Auszeichnungssprachen ausgezeichnet sind, mit Hilfe geeigneter externer Parser einzulesen (siehe Abbildung 6.3).

Ein Beispiel hierfür ist ein im Rahmen dieser Arbeit entwicklter RTF-Parser. Er kann beispielsweise mit Hilfe des Textverarbeitungssystems Microsoft Word erstellte Dokumente parsen

[1] Für Näheres zur ESIS-Form siehe [32].

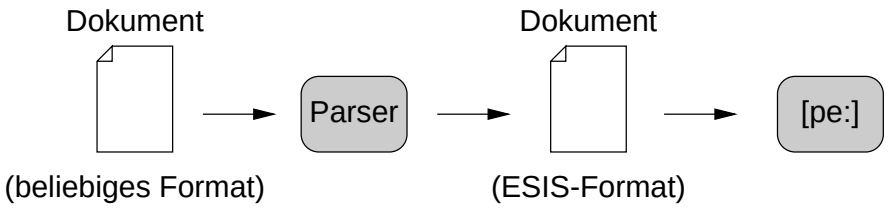

Abbildung 6.3: Verwendung externer Parser im [es:] System: Die Komponente [pe:] liest Dokumente im neutralen ESIS-Format ein und erreicht damit eine Unabhängigkeit von konkreten Formaten, solange diese kein mächtigeres Konzept verwirklichen, als in 2.3.1 vorgestellt wird.

und als Ausgabe ein Dokument im ESIS-Format ausgeben, das derart aufbereitet von [pe:] eingelesen werden kann. Es ist anzumerken, dass die Qualität der Übersetzung eines von einem RTF-Parser verarbeiteten Dokumentes bezüglich der erzielten logischen Auszeichnung stark von dem betroffenen vorgegebenen Dokument abhängt. Dieses ist weitgehend ausschließlich graphisch ausgezeichnet (siehe 2.2.1) und enthält nicht immer alle nötigen Informationen, die nötig sind, um es in ein *vollkommenes* logisch ausgezeichnetes Dokument zu übersetzen.

Als Resultat des Ladevorganges wird in [pe:] ein logischer Baum aufgebaut, dessen logische Objekte logische Attribute besitzen können (siehe 4.3). Die Struktur des logischen Baumes ist im Rahmen einer Designapplikation unveränderlich, es ist aber wohl möglich, den logischen Objekten neue Attribute hinzuzufügen (siehe 4.8.1), die das System jedoch von den ursprünglich aus dem eingelesenen Dokument stammenden logischen Attributen unterscheiden kann. Wichtig ist zu betonen, dass die [pe:] Komponente den *gesamten* logischen Baum einliest und *vollständig* die gesamte Zeit über während einer Designapplikation im Speicher hält. Dies ist bei großen Dokumenten ein sehr speicheraufwendiges Vorgehen, das aus mehreren im Folgenden teilweise aufgeführten Gründen nötig ist.

1. **Komplexe Umgebungsprädikate**
   Vor dem Applizieren von Designregeln wird mit Hilfe des Matching-Algorithmus geprüft, welche der gegebenen Designregeln auf ein logisches Objekt angewendet werden muss. Diese Prüfung kann für ein logisches Objekt, auf das aktuell eine Designregel angewendet werden soll, einen *Blick in die Zukunft* auf noch nicht eingelesene Dokumentbestandteile erfordern.

2. **Multi-View-Konzept**
   Mit Hilfe des Multi-View-Konzeptes ist es möglich, die logischen Objekte eines Dokumentes mehrfach zu verarbeiten (siehe Abbildung 4.23), um z.B. für ein gegebenes Dokument ein Inhaltsverzeichnis und den normalen Textteil zu erzeugen.

3. **Beamer-Konzept**
   Ebenso wie das Multi-View-Konzept erzeugt das Beamer-Konzept einen Applikationspfad, der evtl. mehrfach durch ausgewählte logische Objekte des Dokumentes verlaufen kann.

Einige dieser Gründe würden ein partielles Löschen des logischen Baumes nach Teilverarbeitungsschritten oder eine Auslagerungen von Dokumentteilen auf Platte erlauben. In ihrer Gesamtheit legen sie jedoch nahe, den Weg des vollständigen Einlesens und parallelen Speicherns zu wählen. Diese Entscheidung wird zum einen insbesondere dadurch erleichtert, dass Speicher heute nicht mehr einer der absolut limitierenden Faktoren in einem Rechensystem ist, wie er es noch vor wenigen Jahren war. Zum anderen greift weiterhin das Argument der schnellen Ausführung einer Designapplikation, das insbesondere auch beim Interaktiv-Modus stark an Bedeutung gewinnt.

Einlesen und Verwalten der Designregeln:
Die in einer Designspezifikation enthaltenen [em:] Designregeln werden mit Hilfe einer einfachen proprietären Schnittstelle eingelesen. Sowohl die Formatierungen samt ihrer Parametrisierung als auch die Umgebungsprädikate der Designregeln werden in speziellen Datenstrukturen verwaltet.

Einlesen und Verwalten von Compoundobjekten:
Während alle in [em:] definierten Spezifikationsobjekte in [pe:] bekannt sind und gegebenenfalls sogar eine individuelle Programmierung für sich innerhalb von [pe:] erfordern (alle Spezifikationsobjekte, deren Definition in [em:] eine *aktive Funktionalität* in [pe:] erfordern, wie z.B. die Spezifikationsobjekte `Protocol`, `SysCall`, `Trash` und `Beamer`), müssen die Definitionen für die gemäß [em:] verfügbaren Compoundobjekte aus einer externen Quelle eingelesen werden: Sie können von den Nutzern frei definiert werden.

Designapplikation:
Die Designapplikation wird von der [pe:] Komponente wie in Kapitel 4 erklärt ausgeführt. Dazu wird für einen View `n` der Matching-Algorithmus auf das Wurzel-Objekt des logischen Baumes angewendet und als Resultat daraus (falls möglich) eine geeignete Designregel auf dieses logische Objekt angewendet.

Dies geschieht in [pe:], indem lediglich ein Verweis von dem betroffenen logischen Objekt auf die vom Matching-Algorithmus gewählte Designregel gesetzt wird; die Designregel wird nicht kopiert. Statt dessen wird ein Verwaltungsblock angelegt, mit dessen Hilfe die Werte der Spezifikationsvariablen und Spezifikationsattribute der Formatierung verwaltet werden. Dieses Vorgehen wirkt sich, wie Programmanalysen in einer frühen Entwicklungphase von [pe:] ergeben haben, sehr positiv auf die Speichereffizienz des Systems aus.

Der weitere Ablauf einer Designapplikation wird durch die Zusammenarbeit der in den Designregeln enthaltenen Konnektoren und dem Matching-Algorithmus gesteuert. Dabei ausgewertete Spezifikationsobjekte wie z.B. das `Protocol` und das `SysCall` Spezifikationsobjekt werden von [pe:] unmittelbar umgesetzt und ausgeführt. So wird von [pe:] z.B. beim Vorkommen eines `Protocol` Spezifikationsobjektes dessen gesamter Unterbaum, wie in 4.8.3 spezifiziert, in eine Datei geschrieben.

Interaktiv-Betrieb

Im Interaktiv-Betrieb von [pe:], der der Erstellung einer Designspezifikation dient, ändert sich die Funktionsweise im Vergleich zum Batch-Betrieb von [pe:] in einigen Punkten. Dies ist zunächst der Punkt der Verwaltung und Speicherung von Designregeln.

Verwalten und Speichern von Designregeln:
Im interaktiven Betrieb wird [pe:] dazu verwendet, durch die Interaktion eines Nutzers mit [de:] erstellte und veränderte Designregeln unverzüglich in seinen internen Datenstrukturen zu aktualisieren. Am Ende einer Arbeitssitzung respektive bei Wunsch eines Nutzers müssen die neu erstellten und geänderten Designregeln von [pe:] persistent gespeichert werden.

Speichern von Compoundobjekten:
Im Verlauf einer Arbeitssitzung eines Designers zur Erstellung einer Designspezifikation können durch ihn Compoundobjekte erstellt worden sein, die bei einer späteren Designapplikation verwendet werden und deshalb beim Start von [pe:] eingelesen werden müssen. Aus diesem Grund müssen die im Rahmen des Interaktiv-Betriebes von [pe:] erstellten Compoundobjekte ebenfalls extern persistent gespeichert werden können.

Designapplikation:
Die Designapplikation wird im Interaktiv-Modus vom grundlegenden Ablauf her wie beim Batch-Betrieb ausgeführt. Im Unterschied zum letzteren Modus hier nun aber beliebig oft in Abhängigkeit der Änderungen an der aktuell durch den Nutzer berabeiteten Designspezifikation – jede Änderung der bearbeiteten Designspezifikation macht eine Designapplikation nötig.

Im Unterschied zum Batch-Modus werden im Interaktiv-Modus im Verlauf einer Designapplikation weiterhin zusätzlich gemäß den Vorgaben des Nutzers Auswertungsalgorithmen aktviert, die zusätzliche Datenstrukturen anlegen. Mit Hilfe dieser Datenstrukturen werden Auswirkungen und Querbeziehungen von Spezifikationsvariablen- und Spezifikationsattribut-Wertsetzungen in Spezifikationsobjekten berechnet und protokolliert. Sie können mit Hilfe der [de:] Komponente den Nutzern für eine Auswertung visualisiert werden.

6.1.4 Stabilitätsforderungen

Im Interaktiv-Modus wird die [pe:] Komponente beliebig oft durch Interaktion des Nutzers mit der [de:] Komponente zu Designapplikationen veranlaßt. Beachtet man, dass [pe:] kein fertig entwickeltes Produkt ist, sondern vielmehr ein experimentelles Werkzeug darstellt, mit dessen Hilfe neuartige Konzepte realisiert und getestet werden sollen, bietet diese große Anzahl an Designapplikationen der [pe:] Komponente genügend *Spielraum*, um im Rahmen eines aktuell getesteten Moduls einen Fehler zu verursachen – und als Konsequenz daraus *unkontrolliert* zu terminieren. Dabei würde die in dieser Arbeitssitzung des Designers mit dem dem [es:] System erstellte bzw. seit der letzten Sicherung geänderte Designspezifikation sowie mit ihr *unwiederbringliche*, da spontan getroffene Designentscheidungen, verlorengehen.

Aus diesem Grund ist in [pe:] ein Fehlerbehandlungs-Manager integriert, der Laufzeitfehler des Systems abfängt und sie nach Möglichkeit anzeigt, gegebenfalls korrigiert oder bei unkorrigierbaren Fehlern entsprechende Maßnahmen ergreift. In letzterem Fall werden unter anderem der gegenwärtige Stand der Designspezifikation sowie die aktuell gültigen Compoundobjekt Definitionen in Sicherheitsdateien gespeichert.

6.2 Der Designeditor [de:]

Wie bereits in Abbildung 6.2 dargestellt wurde, ist [de:] technisch betrachtet nichts anderes, als ein interaktives graphisches Interface zu [pe:], mit dessen Hilfe die [pe:] internen Datenstrukturen, die die Designregeln verwalten, manipuliert werden können. Diese indirekte Verwaltung einer Designspezifikation durch einen Editor ist für diese Arbeit ein zentaler Punkt: Sie soll umfassend und überdies benutzerfreundlich, unterstützt werden. Die folgende Erklärung von [de:] soll keine vollständige Nutzeranleitung für den Umgang mit [de:] werden, sondern vielmehr das technische Design von [de:] begründen.

6.2.1 Motivation

Aufgabe eines Designeditors

Die *grundlegende* Aufgabe eines einfachen Designeditors ist es, den Ersteller einer Designspezifikation in die Lage zu versetzen, diese in elektronischer Form speichern zu können, damit sie von einem geeigneten System im Rahmen einer Designapplikation verwendet werden kann. Zu einer qualifizierten und angemessenen Unterstützung gehört aber mehr als nur das bloße Anbieten einer Möglichkeit zur Aufstellung von einzelnen Designregeln, die in ihrer Zusammenfassung eine Designspezifikation bilden. Dies könnte man, überspitzt ausgedrückt, auch mit Hilfe eines normalen Texteditors erledigen.

Es liegt nahe, dass man einzelne Designregeln bei ihrer Erstellung nicht nur isoliert betrachten darf, sondern sie in ihrem Zusammenwirken mit anderen Designregeln sehen *muss*: Designregeln können für gewöhnlich erst gemeinsam in einer Gruppe eine spezifische Aufgabe lösen, z.B. die Festlegung des Designs einer nummerierten Liste.

Aus diesem Grund müssen derartige Zusammenhänge, die zwischen den Designregeln einer Gruppe herrschen, durch die Nutzer, die mit einem Designeditor arbeiten, verwendbar sein. Sie müssen Vorteile bringen und zur ganzheitlichen Erstellung einer *Einheit Designspezifikation* verwendet werden – und nicht nur zur Erstellung einer Menge von einzelnen, lose zusammenhängenden Designregeln. Dies geht aber nur, wenn ein entsprechendes Konzept diese Wechselwirkung zwischen den einzelnen Komponenten, die zu einer Designspezifikation beitragen, auch darstellen und zur Verwendung durch die Designer anbieten kann.

Weiterhin muss ein Designeditor auch das *Debugging* von Designspezifikationen unterstützen: Designspezifikationen sind komplexe Verarbeitungsvorschriften, die von ihrem Wesen her mit *richtigen Programmen* vergleichbar sind. Im Rahmen des Software-Engineerings hat man sich schon lange mit der Einsicht abgefunden, dass selbst bei bestem methodischem Vorgehen Fehler bei der Erstellung von Programmen auftreten, die, verursacht durch einen Menschen oder einer Maschine, von Hand nur schwer zu finden sind, bei geeigneter Werkzeugunterstützung und einer dadurch ermöglichten systematischen Suche hingegen aber oft schnell zu lokalisieren sind.

Die Aufgabe des Designeditors [de:] ist somit, den gesamten Prozess der Erstellung, Wartung, Fehlerbehebung, Übernahme und Weiterentwicklung von Designspezifikationen als ganzheitliche Aufgabe zu unterstützen.

Mögliche Erklärungsansätze für [de:]

In der folgenden Vorstellung des Designeditors [de:] gilt es zu erklären, auf welche Art und Weise er die Konzepte und Objekte der Designspezifikationsmethodik [em:] unter Beachtung der Anforderung, den gesamten Prozess der Designspezifikationserstellung als ganzheitliche Aufgabe zu unterstützen, realisiert. Dazu gibt es zwei Erklärungsansätze, die diese Aufgabe von zwei gegensätzlichen Richtungen aus angehen:

1. **Funktionale Betrachtung**
   Bei diesem Erklärungsansatz des Designeditors [de:] wird ausgehend von der Designspezifikationsmethodik [em:] untersucht, wie die dort eingeführten Konzepte und Objekte in [de:] umgesetzt sind. Der Leitfaden bei dieser Art des Vorgehens ist somit:
   Was von [em:] ist wie in [de:] realisiert?
   Er bedingt ein passives Einführungsmodell von [de:], bei dem von [em:] aus Bezug auf [de:] genommen wird.

2. **[de:] by example**
   Dieser zweite Erklärungsansatz ist vom Konzept her invers zu dem oben vorgestellten. Sein Leitsatz ist:
   Wie arbeitet man mit [de:]?
   Er stellt ausgehend von einem exemplarischen Arbeiten mit [de:] vor, was man mit der Designspezifikationsmethodik [em:] erreichen kann und ist somit ein Ansatz, der aktiv [de:] in den Mittelpunkt stellt.

Während der erste Ansatz erfordert, dass man [em:] inhaltlich vor der Vorstellung von [de:] vollständig kennt und verinnerlicht hat, ist der zweite Erklärungsansatz diesbezüglich von einer einfacheren Natur. Er stellt die Möglichkeit zur Verfügung, durch die Beschreibung einer Benutzungsoberfläche unter Verwendung einfacher exemplarischer Beispiele spielerisch sowohl in [de:] als auch in [em:] *einzusteigen*. Er kommt dem Menschen als *Homo ludens* entgegen und entspricht der heutigen Erwartungshaltung von Nutzern beim Erlernen neuer Software und den ihr zugrundeliegenden Konzepten: Ein unverzügliches produktives Arbeiten mit der neuen Software muss vom ersten Moment an möglich sein, ohne vorher lange Literatur studieren zu müssen. Für einen vertieften Einblick und die ausschöpfende Nutzung der Designspezifikationsmethodik [em:] ist es aber trotzdem nötig, Rückgriff auf Kapitel 4 zu nehmen.

Der zweite Ansatz der Erklärung über Beispiele bringt weiterhin neben der Einführung der Benutzungsoberfläche als Nebeneffekt mit sich, eine zweite alternative Erklärung von [em:] zu realisieren, die im Vergleich zu der in Kapitel 4 gegebenen wissenschaftlich und technisch exakten Vorgehensweise einfacher zu verstehen ist, indem sie einfach die Ergebnisse von Kapitel 4 anwendungsorientiert und komprimiert darstellt.

Nach Abwägung der oben genannten Gründe wird in der folgenden Erklärung des Designeditors [de:] die zweite vorgestellte Vorgehensweise *[de:] by example* gewählt, da der erste Ansatz eher für die Realisierung eines Nachschlagewerkes geeignet ist, mit dessen Hilfe man gezielt bei der Arbeit mit [de:] auftretende Problemstellungen behandeln kann. Somit wird

im Folgenden im Rahmen der Einführung von [de:] gemäß einem von Kapitel 4 unabhängigen Leitfaden die Designspezifikationsmethodik [em:] in Form einer *praxisnahen* Darstellung ein zweites mal behandelt. Die grundlegenden Ideen und Konzepte von [em:] werden jedoch nicht noch einmal erarbeitet. Für ein tiefgreifendes Verständnis von ihnen ist die Lektüre von Kapitel 4 unabdingbar.

6.2.2 Spezifikationsobjekte als *Bausteine* für Formatierungen

Spezifikationsobjekte und ihre Darstellung

Spezifikationsobjekte sind *die* Konstrukte, mit deren Hilfe im Rahmen von Formatierungen für logische Objekte in [de:] die Art und Weise der Verarbeitung der logischen Objekte eines Dokumentes spezifiziert werden kann. Von ihrer Wirkung her können die in [de:] verfügbaren Spezifikationsobjekte grob in drei Klassen eingeteilt werden:

- Dies ist zunächst die Klasse der *Layoutobjekte*, mit deren Hilfe die Nutzer die Anwendung ihnen vertrauter Layouteffekte wie Zeilenumbruch, Glyphauswahl oder Anzeige am Bildschirm spezifizieren können.
- Weiterhin sind dies die *Auswertungsobjekte*, mit deren Hilfe z.B. Feinheiten bei der Anwendung von Layouteffekten (z.B. Setzung der Schriftart für den Text innerhalb einer Hervorhebung) gezielt gesteuert werden können.
- Schließlich gibt es die Klasse der *Hilfsobjekte*, mit deren Hilfe beispielsweise die Interaktion eines verarbeiteten logischen Objektes mit seiner Umgebung (z.B. Speicherung in einer Datei) gesteuert werden kann.

Betrachtet man die Spezifikationsobjekte näher, so verkörpern diese Funktionalitäten, wie man sie auch in der realen Welt findet. So gibt es in [de:] beispielsweise ein `Window` Layoutobjekt zur Vorgabe, dass ein verarbeitetes logisches Objekt ein Fenster am Bildschirm erzeugen soll, sein `Scroll` Layoutobjekt zur Vorgabe, dass ein scrollbarer Bereich erzeugt werden soll, ein `Vertical` Layoutobjekt zur Vorgabe, dass für das verarbeitete logische Objekt sein Inhalt vertikal untereinander angeordnet werden soll und ein `Protocol` Spezifikationsobjekt zur Vorgabe, dass die für ein logisches Objekt und seine Kinder gewählte Verarbeitung in eine Datei *protokolliert* werden soll.

Da in der realen Welt beliebige Funktionen über Parameter gesteuert werden können, wird dies auch in den Spezifikationsobjekten von [de:] modelliert. Die zur Steuerung von Spezifikationsobjekten verwendeten Attribute heißen Spezifikationsattribute, im Falle der Layoutobjekte können sie auch Layoutattribute genannt werden. Jedes Spezifikationsobjekt besitzt einen eigenen Satz von Attributen (der unter Umständen leer sein kann), die von den Nutzern der Verwendung entsprechend gesetzt werden können (siehe hierzu 6.2.7, Objekteigenschaften, auf Seite 216). Ein `Window` Layoutobjekt besitzt beispielsweise die Layoutattribute `height` und `width` zur Spezifizierung seiner Ausmaße.

Auswertungsobjekte können zusätzlich zu den Spezifikationsattributen noch Spezifikationsvariablen besitzen. Diese können, wie sich später noch im Detail zeigen wird, zur gezielten Setzung der Werte von Spezifikationsattributen verwendet werden. Ein typisches Beispiel hierzu ist die Möglichkeit, die Schriftart für den gesamten Text, der in einem logischen Objekt `Hervorhebung` vorkommt, in einem speziellen Font zu setzen.

Spezifikationsobjekte werden wegen ihrer Verkörperung von realen Objekten in [de:] als *Bausteine*, wie sie auch in Baukästen vorhanden sind, betrachtet und haben demgemäß verschiedene äußere Formen. In [de:] werden die im Folgenden vorgestellten drei Formen unterschieden, die insbesondere für die Klassifizierung von Layoutobjekten geeignet sind, jedoch sinngemäß auch für die restlichen Spezifikationsobjekte angewendet werden können:

- **Zielobjekte** sind Bausteine, die ein physikalisches Medium darstellen. Hierunter fällt z.B. das `Window` Layoutobjekt. Graphisch werden Zielobjekte in [de:] durch Objekte der in Abbildung 6.4 a dargestellten Form repräsentiert und zur Identifizierung der durch sie verkörperten Funktionalität mit dem Namen der entsprechenden Layoutobjektklasse ergänzt (als Beispiel hierzu siehe Abbildung 6.4 d).

- **Transiente Objekte** sind Bausteine, die die Transformation und andere Verarbeitungen von Objekten verkörpern. So spezifiziert z.B. das `Vertical` Layoutobjekt eine Verarbeitung von Objekten in der Form, dass sie vertikal untereinandet angeordnet werden und das `Protocol` Spezifikationsobjekt die Protokollierung einer für logische Objekte gewählten Verarbeitung in eine Datei. Transiente Objekte werden graphisch in [de:] wie in Abbildung 6.4 b und individualisiert wie in Abbildung 6.4 e dargestellt.

- **Quellobjekte** sind Bausteine, die die Erzeugung von durch andere Spezifikationsobjekte zu verarbeitende Objekte verkörpern. Ein typisches Quellobjekt ist das `Glyphs` Layoutobjekt, das für ein logisches Objekt Buchstaben erzeugt sowie das `Picture` Layoutobjekt, das für ein logisches Objekt eine Graphik aus einer Datei lädt. Quellobjekte werden allgemein wie in Abbildung 6.4 c und individualisiert wie in Abbildung 6.4 f dargestellt.

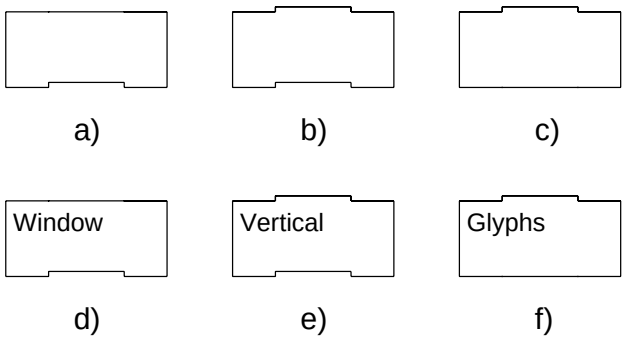

Abbildung 6.4: Objektformen von Spezifikationsobjekten: In der oberen Reihe sind die allgemeinen Formen der Quell-, transienten und Zielobjekte dargestellt, in der unteren Reihe in entsprechender Reihenfolge konkrete Ausprägungen dieser Klassen.

Die Spezifikationsattribute, die ein Spezifikationsobjekt in [de:] besitzt, sind nicht anhand der graphischen Darstellung, die für ein Spezifikationsobjekt gewählt wird, zu erkennen. Somit ist mit Hilfe der in Abbildung 6.4 in der unteren Reihe dargestellten Objekte *nur* das prinzipielle Verhalten eines Spezifikationsobjektes zu erkennen, das dieses aber bereits im Wesentlichen beschreibt.

Die sparsame Verwendung von Farben für die Darstellung der in [de:] vorkommenden Spezifikationsobjekte kann neben der Form und der Namensanzeige von Spezifikationsobjekten den Designern dabei helfen, die Aufgabe von Spezifikationsobjekten in Formatierungen schnell im Überblick zu erkennen. Aus diesem Grund werden die Spezifikationsobjekte gemäß den drei Grobklassen Layoutobjekte, Auswertungsobjekte und Hilfsobjekte in [de:] mit jeweils einer gemeinsamen Füllfarbe dargestellt.

Die zur Darstellung der in [de:] vorkommenden Spezifikationsobjekte verwendeten Farben sind neben dem Basiskriterium der Klassenzugehörigkeit weiterhin von dem Zustand abhängig, in dem sich ein Spezifikationsobjekt aktuell beim Zeitpunkt seiner Darstellung befindet. Die möglichen Zustände, in denen sich ein Objekt in einer *einfachen* Darstellung[2] befinden kann sowie ihre Aufgabe im Rahmen der Interaktion des Systems mit den Nutzern, sind in folgender Übersicht zusammengestellt:

- **Normaler Zustand:**
  Grundzustand eines Spezifikationsobjektes, der durch die Verwendung der Grundfarbe eines Objektes seine Zugehörigkeit zu einer der drei möglichen Spezifikationsobjektklassen ausdrückt.

- **Vorselektierter Zustand:**
  In diesem Zustand drückt ein Spezifikationsobjekt seine Bereitschaft aus, dass auf ihm eine Aktion ausgeführt werden kann. Die Farbgebung sollte zur Grundfarbe eines Spezifikationsobjektes ähnlich sein (z.B. eine Abdunklung des Farbtones oder Schraffierung), um nach wie vor die Zugehörigkeit zur Spezifikationsobjektklasse anzuzeigen.

- **Selektierter Zustand:**
  Dieser Zustand zeigt an, dass das Spezifikationsobjekt *das* im System momentan selektierte Spezifikationsobjekt ist[3]. Dies drückt aus, dass spezielle Kontrollfenster, die in [de:] zur Verfügung stehen, die Werte für dieses Spezifikationsobjekt zur Verarbeitung beinhalten.

- **Vorselektiert/selektierter Zustand:**
  Dieser Zustand ist eine Kombination der Eigenschaften der zwei vorausgehenden Zustände.

Das im Designeditor [de:] vorgegebene Farbschema (für Spezifikationsobjekte und alle anderen graphischen Objekte) ist sorgfältig ausgewählt worden und wird automatisch als *Default*-Einstellung vorgegeben. Die Farben für Spezifikationsobjekte sowie für die ebenfalls im System

[2] *Mehrfache* Darstellungszustände eines Spezifikationsobjektes werden in Abschnitt 6.2.6 behandelt.
[3] Zu jedem Zeitpunkt ist in [de:] höchstens ein Objekt selektiert.

realisierten logischen Objekte sind aber trotzdem über eine Konfigurationsdatei frei festlegbar. Änderungen hierbei sollten aber mit Vorsicht vorgenommen werden.

Kombinationen von Spezifikationsobjekten

Aufgrund der äußeren Form der Bausteine *Spezifikationsobjekt* in [de:] ergeben sich wie in realen Baukästen *sinnvolle* Kombinationsmöglichkeiten, gemäß denen Spezifikationsobjekte in Formatierungen angeordnet werden können. Die Ein- und Ausbuchtungen der Spezifikationsobjekte, die auch Konnektoren genannt werden, legen dabei zunächst die direkten Kombinationen Ziel- und transientes Objekt (siehe Abbildung 6.5 a), transientes und Quellobjekt (siehe Abbildung 6.5 b), sowie Ziel- und Quellobjekt (siehe Abbildung 6.5 c) nahe: Bei ihnen passen die Konnektoren der beteiligten Spezifikationsobjekte direkt zusammen. Unmittelbar auf diesen exakt zwei Spezifikationsobjekte umfassenden Kombinationen lassen sich dann größere Formatierungen erstellen, die mehr als nur zwei Spezifikationsobjekte enthalten.

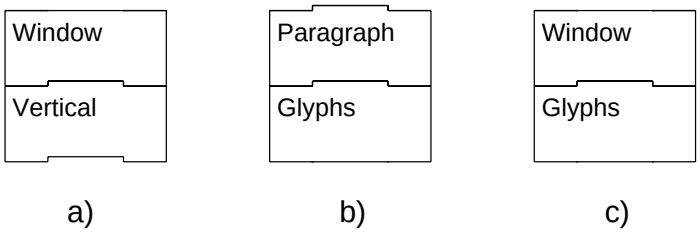

Abbildung 6.5: Einfache Spezifikationsobjekt-Kombinationen: In dieser Abbildung sind die direkten offensichtlichen Kombinationsmöglichkeiten der drei Formen von Spezifikationsobjekten mit Hilfe konkreter Spezifikationsobjekte dargestellt.

Die Bedeutung derartiger Spezifikationsobjekt Anordnungen in Formatierungen lässt sich einfach induktiv unter Abstützung auf die einzelnen beteiligten Spezifikationsobjekte folgern. Verdeutlicht wird dies in einer Formatierung, die in Abbildung 6.6 dargestellt ist und eine vertikale Anordnung der Spezifikationsobjekte `Window`, `Scroll` und `Vertical` ist. Sie spezifiziert eine Verarbeitung, die, angewendet auf ein logisches Objekt, für dieses ein Fenster am Bildschirm erzeugt, das in sich einen scrollbaren Bereich enthält, in dem wiederum ein vertikaler Anordner lokalisiert ist.

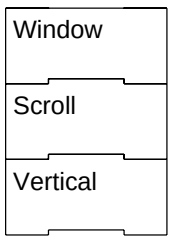

Abbildung 6.6: Größere Spezifikationsobjekt-Kombination

Neben diesen offensichtlichen Kombinationsmöglichkeiten des direkten vertikalen Zusammensetzens von Spezifikationsobjekten in Formatierungen gibt es in [de:] noch eine weite-

re Möglichkeit, Spezifikationsobjekte zu kombinieren: Horizontal. Ein möglicher Fall hierzu ist in Abbildung 6.7 a dargestellt. Hier sind horizontal links und rechts neben dem zentralen `Scroll` Layoutobjekt noch zwei weitere `Scroll` Layoutobjekte angebracht. Die Bedeutung dieser horizontalen Anordnung wird in Abbildung 6.7 b verdeutlicht: Die neben dem zentral angeordneten `Scroll` Layoutobjekt angebrachten zwei `Scroll` Layoutobjekte sind *gleichwertige* Geschwister zu diesem. Die in [de:] benutzte linke Darstellung gleicht zum einen auf naheliegende Weise die fehlende Möglichkeit aus, unterhalb eines Spezifikationsobjektes mehrere andere Spezifikationsobjekte anzufügen. Zum anderen ermöglicht sie eine kompakte Darstellung (in vertikaler Richtung) von Formatierungen.

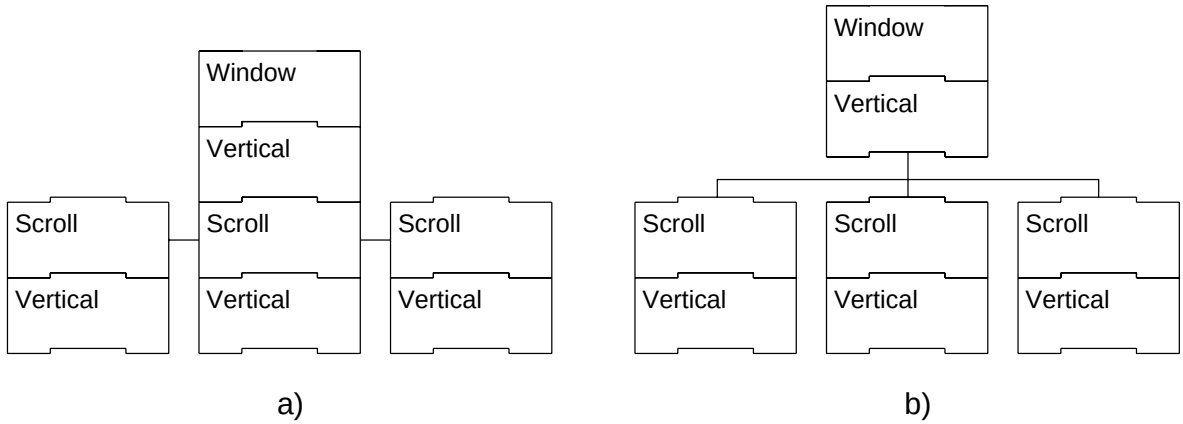

Abbildung 6.7: Komplexe Spezifikationsobjekt-Kombination: In komplexen Fällen können Spezifikationsobjekte auch horizontal nebeneinander angeordnet werden. Die Erklärung der Bedeutung einer derartigen Anordnung ergibt sich durch die Struktur auf der rechten Seite der Anordnung, in der Verbindungslinien zur Verdeutlichung der strukturellen Anordnung verwendet worden sind: Die horizontal angeordneten Spezifikationsobjekte sind gleichberechtigte Geschwister.

An dieser Stelle ist zu erwähnen, dass bei den einfachen vertikalen Kombinationsmöglichkeiten von Spezifikationsobjekten, von denen ein Ausschnitt in Abbildung 6.5 dargestellt ist, auch alle anderen Möglichkeiten, die nicht explizit aufgeführt worden sind, möglich und in [de:] erlaubt sind. Die Bedeutung dieser Kombinationen ist in Abschnitt 4.2.4 auf Seite 101 erklärt.

Vorteile der Spezifikationsobjekte als graphisch visualisierte Bausteine

Die Vorteile der in [de:] als Bausteine zur Erstellung von Formatierungen eingeführten Spezifikationsobjekte wirken sich in zwei verschiedenen Bereichen aus. Dies ist zum einen der funktionale Bereich, der entscheidet, was man mit Hilfe einer Formatierung erreichen kann. Zum anderen ist dies der graphische Bereich, der entscheidet, wie einfach eine Formatierung von einem Designer zu erstellen und zu verstehen ist.

Im Rahmen des funktionellen Bereiches ist festzuhalten, dass die Menge der gegebenen Spezifikationsobjekte bei geeignetem Aufbau zu sehr flexiblen Formatierungen führt, die derart

bisher nicht spezifizierbar waren. Einer der bedeutendsten Effekte, der hervorzuheben ist, ist die beliebige Kombinierbarkeit von Spezifikationsobjekten, um beliebige Layouteffekte erzielen zu können. Als Maßstab zum Aufbau eines Pools von Spezifikationsobjekten, die man zum Aufbau benötigter Formatierungen in einer spezifischen Anwendung braucht, ist in 4.2.1 systematisch eine Menge von Kriterien an eine Spezifikationsobjekt-Basismenge aufgestellt worden.

Die Vorteile auf Grund der graphischen Darstellung von Spezifikationsobjekten sowie deren Kombinationen in Form von Formatierungen sind mannigfaltig und an dieser Stelle noch nicht alle zu erkennen. Hier kann jedoch schon behauptet werden, dass eine komplexe Formatierung, wie sie z.B. in Abbildung 6.7 a dargestellt ist, unmittelbar intuitiv und somit wesentlich einfacher zu verstehen ist, als dies in einer textuellen Form möglich wäre.

6.2.3 Das Erstellen von Formatierungen

Formatierungen beschreiben in lokaler Form die Auswirkungen, die ein logisches Objekt zum Layout eines Dokumentes beiträgt. Sie bestehen, wie bereits in 6.2.2 dargestellt wurde, aus einer beliebigen hierarchischen Kombination von Spezifikationsobjekten. Der Aufbau von Formatierungen und somit das Einfügen, Löschen, Verschieben und Kopieren von Spezifikationsobjekten in Formatierungen wird in [de:] graphisch interaktiv durch *Drag and Drop* Operationen ausgeführt.

Obgleich die Erstellung von Formatierungen in [de:] mit Hilfe von zwei unabhängigen Fenstern (Hauptfenster und Formatierungsfenster) möglich ist, die unterschiedliche Sichten auf die Formatierungen bieten, ist in beiden Fällen die im Folgenden besprochene graphische Interaktion der Nutzer mit dem System weitgehend identisch: Aus dem Pool der zur Verfügung stehenden Spezifikationsobjekte können beliebig viele Instanzen der benötigten Spezifikationsobjekte entnommen und zum Aufbau einer Formatierung verwendet werden. In einer Formatierung verbaute Spezifikationsobjekte können dort wieder gelöscht, an eine andere Stelle verschoben oder auch identisch an eine andere Stelle kopiert werden. Hiervon abweichende Vorgehensweisen werden lokal an den entsprechenden Stellen der Vorstellung von [de:] besprochen.

Logische Objekte in Formatierungen

Ausgangspunkt bei der Erstellung einer Formatierung für ein logisches Objekt ist immer dieses logische Objekt, für das eine Formatierung erstellt werden soll. Es trägt selbst zur Formatierung nichts bei, zeigt durch seine Einbeziehung aber an, dass die Formatierung für das logische Objekt selbst aufgestellt wird. Aus diesem Grund müssen die logischen Objekte passend zur visuellen Darstellung der Spezifikationsobjekte in [de:] modelliert werden. Die in [de:] getroffene Entscheidung hierzu ist in Abbildung 6.8 dargestellt.

Für die logischen Objekte wird ebenso wie für die Spezifikationsobjekte in [de:] eine Füllfarbe zur graphischen Hervorhebung verwendet, um somit insgesamt eine gute Unterscheidbarkeit aller in [de:] vorkommenden Objekte untereinander zu erreichen. Hierzu wird wie bei den Spezifikationsobjekten eine Basisfarbe gewählt, die in Abhängigkeit des Zustandes eines logischen Objektes verschiedene Ausprägungen annehmen kann. Die möglichen Zustände sind

Abbildung 6.8: Modellierung der logischen Objekte in [de:]: In der Abbildung ist ein modelliertes logisches Objekt `document` dargestellt. Die logischen Attribute, die es besitzt, werden in dieser Darstellung nicht visualisiert.

identisch zu den Zuständen der Spezifikationsobjekte, die auf Seite 187 eingeführt worden sind und können systematisch von den Erklärungen her übertragen werden.

Spezifikationsobjekte in Formatierungen

Spezifikationsobjekte sind die elementaren Bauteile in [de:], mit deren Hilfe Formatierungen aufgebaut werden. Im Folgenden werden die einzelnen Operationen der direkten Manipulation beschrieben, mit deren Hilfe der Aufbau von Fomatierungen erfolgt.

Einfügen von Spezifikationsobjekten:
Um ein neues Spezifikationsobjekt in eine Formatierung einzufügen, wird das gewünschte Spezifikationsobjekt im Pool der Spezifikationsobjekte mit der linken Maustaste genommen (Druck auf die linke Maustaste) und in die Formatierung gezogen (*Drag-Operation*). Wird das gezogene Spezifikationsobjekt dabei in die Nähe eines anderen Objektes gebracht (logisches Objekt oder Spezifikationsobjekt), an das es mit Hilfe einer *Drop-Operation* angefügt werden kann, zeigt dieses Objekt (im Folgenden *Drag-Ziel* genannt) die Ausführbarkeit dieser Aktion durch eine Zustandsänderung (Vorselektierter Zustand), sowie in mehrdeutigen Fällen zusätzlich durch einen farbigen Balken an einem der Ränder des Objektes, an. Wird nun das gezogene Spezifikationsobjekt *fallengelassen*, wird es an das Drag-Ziel angefügt. Wird das gezogene Objekt hingegen über einem freien Bereich fallengelassen, verschwindet es, ohne eine Veränderung der Formatierung zu bewirken.

Ein Spezifikationsobjekt kann auf verschiedene Weise an ein vorhandenes Objekt angefügt werden, je nachdem, an welcher Stelle das neue Spezifikationsobjekt fallengelassen wird. Ist das Drag-Ziel ein logisches Objekt, so wird das hinzuzufügende Spezifikationsobjekt in allen Fällen unterhalb des logischen Objektes angefügt (siehe Abbildung 6.9 a). Ist das Drag-Ziel hingegen ein Spezifikationsobjekt, gibt es vier mögliche Stellen, an denen das neue Spezifikationsobjekt angefügt werden kann:

- Wird das Spezifikationsobjekt oberhalb bzw. unterhalb des Drag-Ziels fallengelassen, so wird es oberhalb bzw. unterhalb des Drag-Ziels eingefügt (für Einfügung unterhalb siehe Abbildung 6.9 b).

- Wird das Spezifikationsobjekt links bzw. rechts vom Drag-Ziel fallengelassen, wird es als linkes bzw. rechtes Geschwister des Drag-Ziels eingefügt (für Einfügung links siehe Abbildung 6.9 c).

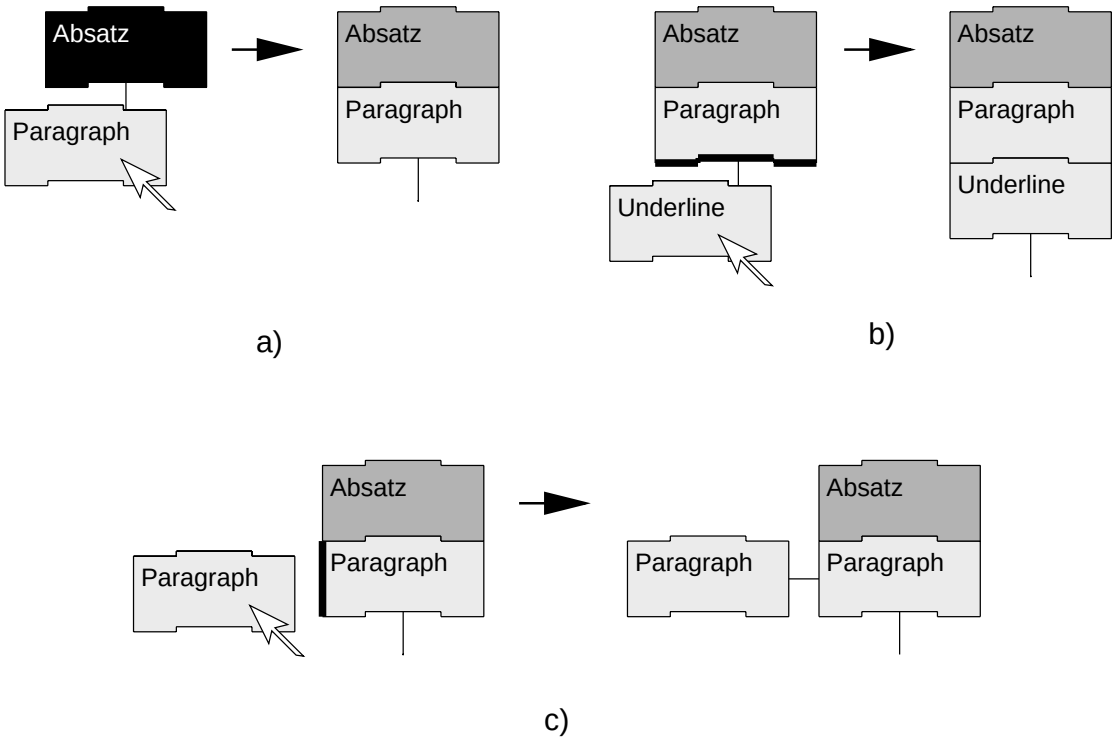

Abbildung 6.9: Drag and Drop Operation von Spezifikationsobjekten

Löschen von Spezifikationsobjekten:
Ein Spezifikationsobjekt wird aus einer Formatierung entfernt, indem es mit der Maus genommen und an einer *leeren* Stelle neben der Formatierung fallengelassen wird. Falls das gelöschte Spezifikationsobjekt in seiner Formatierung einen Vater und Kinder hatte, werden die Kinder zu den Kindern das Vaters. Falls es ein linkes und ein rechtes Geschwister hatte, sind diese nach der Löschung des Spezifikationsobjektes benachbart. Ein Spezifikationsobjekt, das Geschwister hat, kann jedoch nicht gelöscht werden, wenn es entweder Kinder hat oder sich vertikal direkt unterhalb des logischen Objektes befindet, da dann eine eindeutige Rekombination der übrigen Formatierungsteile nicht möglich wäre. In diesem Fall muss von dem Nutzer vor dem Löschen des Spezifikationsobjektes die Formatierung so umgebaut werden, dass obige Bedingung nicht mehr greift. Logische Objekte können in [de:] nicht gelöscht werden.

Verschieben von Spezifikationsobjekten:
Falls ein Spezifikationsobjekt aus einer Formatierung gelöscht werden darf, kann es stattdessen auch an eine andere Stelle in einer Formatierung verschoben werden. Wird ein Spezifikationsobjekt aus der Formatierung aufgehoben und über einem anderen Objekt fallengelassen, entspricht das Ergebnis einer Kombination von Löschen und Einfügen wie oben beschrieben.

Kopieren von Spezifikationsobjekten:
Ein beliebiges Spezifikationsobjekt aus einer Formatierung kann an eine beliebige andere zulässige Stelle in einer Formatierung kopiert werden, indem vor dem Aufnehmen des Objektes mit der linken Maustaste zusätzlich die rechte Maustaste gedrückt wird.

Selektieren von Objekten:
Durch einfaches Anklicken (linke Maustaste drücken und wieder loslassen) eines logischen Objektes oder eines Spezifikationsobjektes mit der linken Maustatste ist es möglich, dieses als aktives Objekt zu selektieren und entsprechend vom System markieren zu lassen. Da in [de:] immer höchstens ein Objekt ausgewählt sein kann, wird dabei gegebenenfalls ein bereits selektiertes Objekt zuvor deselektiert. Wie auf Seite 187 dargestellt wurde zeigt ein Objekt durch seine Farbe an, in welchem Zustand es sich gegenwärtig befindet.

Auf einem ausgewählten Objekt können je nach Art des Objektes verschiedene Operationen ausgeführt werden. Dies ist z.B. bei den Spezifikationsobjekten die Festlegung von Werten für Spezifikationsattribute. Details hierzu finden sich in den folgenden Abschnitten.

`String` Spezifikationsobjekte in Formatierungen

Das `String` Spezifikationsobjekt (siehe 4.4.5 auf Seite 127) ist ein spezielles Spezifikationsobjekt, das anstelle eines `Glyphs` Quellobjektes verwendet werden kann, um z.B. einen String von logischen Zeichen als Folge von graphischen Zeichen zu formatieren. Im Gegensatz zum `Glyphs` Spezifikationsobjekt erlaubt ein `String` Spezifikationsobjekt Attributwerte der graphischen Zeichen wie Schriftart oder -farbe individuell für jedes einzelne Zeichen festzulegen. Es ermöglicht es sogar, jedem logischen Zeichen eine völlig andere Formatierung zuzuweisen, die nicht notwendigerweise ein graphisches Zeichen sein muss, sondern beliebig aus anderen Spezifikationsobjekten zusammengesetzt sein kann.

Da ein `String` Spezifikationsobjekt über eine komplexere Funktionalität als die bisher vorgestellten Spezifikationsobjekte verfügt, sieht es zur Steuerbarkeit dieser Funktionalität über eine graphische Schnittstelle auch anders als die bisher vorgestellten Spezifikationsobjekte aus – und ist folglich auch anders interaktiv zu bearbeiten. Dies wird im Folgenden beschrieben.

Ein `String` Spezifikationsobjekt verfügt über eine Menge von internen Eingängen (siehe Abbildung 6.10 a), an die jeweils eine beliebige *interne Formatierung* (siehe Abbildung 6.10 b) angeschlossen werden kann. Diese internen Formatierungen können wie normale Formatierungen bearbeitet werden. Das erste Spezifikationsobjekt einer internen Formatierung muss an einem der internen Eingänge abgelegt werden. Zur Anzeige der Akzeptanz dieses Vorganges wird das `String` Spezifikationsobjekt im vorselektierten Zustand angezeigt. Soll einem `String` Spezifikationsobjekt ein externer Sohn angefügt werden, muss das gezogene Objekt an der Unterkante des Rahmens des `String` Spezifikationsobjektes fallengelassen werden.

Die Anzahl der internen Eingänge und die Bedingungen (siehe 4.4.5) für die `String` Spezifikationsobjekte lassen sich in einem speziellen Einstellfenster (siehe 6.2.7, Menüeintrag `Windows - String Input`) vorgeben.

6.2.4 Das Formatierungs-Fenster

Das Formatierungs-Fenster ist das erste von zwei Fenstern, mit dessen Hilfe in [de:] Formatierungen bearbeitet werden können. Ein Screen-Shot dieses Fensters ist in Abbildung 6.11 zu sehen.

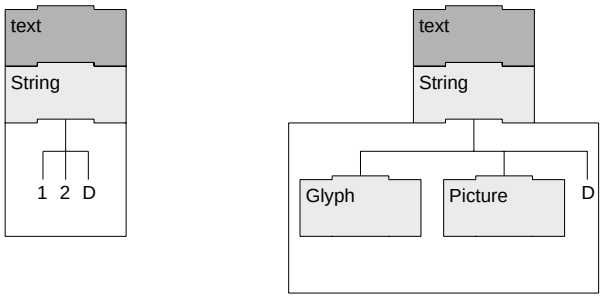

Abbildung 6.10: `String` Spezifikationsobjekt: Die internen Eingänge (drei in obiger Abbildung) dienen zum *Anhängen* der internen Formatierungen. Unter a) ist ein `String` Spezifikationsobjekt in einem Zustand ohne interne Formatierungen dargestellt, in b) mit zwei internen Formatierungen.

Das Formatierungs-Fenster ist horizontal aufgrund einer funktionalen Unterscheidung in zwei Bereiche aufgeteilt. Links sind die verfügbaren Spezifikationsobjekte und rechts die bereits aufgestellten Formatierungen zu sehen und zu bearbeiten:

- **Verfügbare Spezifikationsobjekte:** Im linken Bereich sind alle verfügbaren Spezifikationsobjekte abgelegt. Aus Gründen der Übersichtlichkeit ist der linke Bereich vertikal gemäß der Unterteilung der Spezifikationsobjekte auf Seite 185 in die vier Unterbereiche Layoutobjekte (`Layout`), Auswertungsobjekte (`Evaluation`) und Hilfsobjekte (`Special`) sowie weiterhin die Compoundobjekte (`Compounds`) unterteilt.
- **Aufgestellte Formatierungen** Der rechte größere Bereich stellt in einer tabellenartigen Form alle Formatierungen dar, die bisher aufgestellt wurden.

Mit Hilfe der in Abschnitt 6.2.3 beschrieben Methoden der direkten Manipulation können in dem Formatierungs-Fenster die Formatierungen bearbeitet werden. Dazu ist es unter Umständen nötig, die Darstellungen in den einzelnen Bereichen zu manipulieren. Hierzu stehen für alle Bereiche die im Folgenden aufgeführten Methoden zur Verfügung (bei Eingaben über die Tastatur ist immer derjenige Bereich betroffen, in dem sich gerade der Mauszeiger befindet):

- **Scrollen mit der Maus:** Alle Bereiche lassen sich in Echtzeit scrollen, indem sie bei gedrückter mittlerer Maustaste *gezogen* werden. In gewissen Fällen kann dabei eine der beiden möglichen Dimensionen Horizontal oder Vertikal aufgrund fehlenden Bedarfs deaktiviert sein.
- **Scrollen in kleinen Schritten:** Mit Hilfe der Cursortasten `links`, `rechts`, `oben` und `unten` kann die Darstellung des betroffenen Bereichs in kleinen Schritten jeweils in die entsprechende Richtung verschoben werden.
- **Scrollen an die Ränder:** Die in einem Bereich angezeigte Darstellung kann mit Hilfe der Tasten `l`, `r`, `t` und `b` bündig jeweils an den linken, rechten, oberen oder unteren Rand des Bereiches verschoben werden. Die Tasten `Pos1` und `Ende` verschieben die Darstellung eines Bereiches nach links oben bzw. rechts unten.

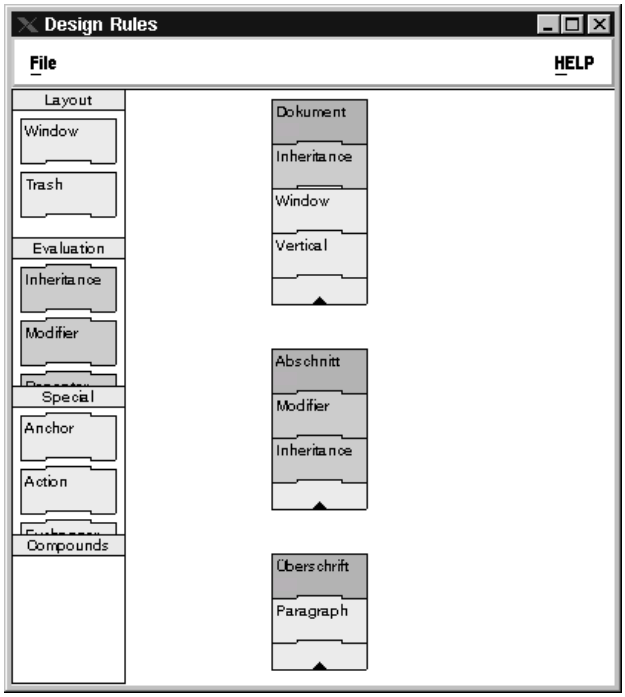

Abbildung 6.11: Formatierungs-Fenster

- **Zentrieren:** Die Darstellung eines Bereiches kann mit Hilfe der Tasten `h` und `v` horizontal bzw. vertikal zentriert werden, sowie mit Hilfe der Taste `c` gleichzeitig in beiden Dimensionen.

- **Zoomen:** Mit Hilfe der Tasten `z` und `Z`[4] kann die Ansicht, die in einem Bereich auf eine Darstellung gewährt wird, verkleinert (`z`) und vergrößert (`Z`) werden.

- **Maximieren:** Die Taste `f` bewirkt eine Skalierung der in einem Bereich angezeigten Darstellung derart, dass sie in maximaler Größe den verfügbaren Bereich ausfüllt.

- **Rücksetzen:** Mit Hilfe der Taste `R` kann die Darstellung in einem Bereich auf Grundwerte zurückgesetzt werden. Dies entspricht einem Zoomfaktor von 1 und einer zentrierten Darstellung.

Bei allen oben aufgeführten Manipulationsmöglichkeiten der Darstellung in einem Bereich erfolgen notwendige Verschiebungen immer *animiert*: D.h., es wird die Anzeige nicht von einer Situation auf die nächste schlagartig umgeschaltet, sondern nötigenfalls mit *dazwischenliegenden* Anzeigen für die Nutzer des Systems wie in einem Zeichentrickfilm vollzogen.

[4]Auf einer Tastatur befindet sich immer nur ein abstraktes Zeichen, das vom Rechensystem interpretiert wird. Insofern bezeichnen im Folgenden Kleinbuchstaben immer das alleinige Drücken der entsprechenden Taste und Großbuchstaben immer das gemeinsame Drücken der Kombination aus der Shifttaste und der entsprechenden Taste.

6.2.5 *Design by Example*

[de:] ist ein interaktiv graphisches Werzeug zur Erstellung von Designspezifikationen gemäß der [em:] Designspezifikationsmethodik, das auf obiger graphischer Umsetzung der Spezifikationsobjekte sowie den oben eingeführten Methoden der direkten Manipulation von Formatierungen beruht. Die bisher vorgestellte Vorgehensweise zur Erstellung von Formatierungen im Formatierungs-Fenster unterstützt dabei bisher aber lediglich den Zusammenbau einzelner isolierter Formatierungen. Dies bringt zwar im Vergleich zur textuellen Aufschreibung von Formatierungen bereits enorme Vorteile mit sich, da durch die im Rahmen einer graphischen Darstellung einer Formatierung vorgenommene Abstraktion für die Nutzer schneller *begreifbar* wird, welche grundlegende Verarbeitung durch eine Formatierung spezifiziert wird.

Dieses Vorgehen beachtet jedoch nicht die Tatsache, dass einzelne Formatierungen nicht losgelöst und vollkommen alleine das Layout von logischen Objekten beschreiben. Vielmehr beschreiben immer zusammengehörige Gruppen von Formatierungen gemeinsam gewisse Verarbeitungsaspekte von Dokumenten, so z.B. die Designbeschreibung einer in einem Dokument vorkommenden Liste (siehe als Beispiel hierzu Abbildung 4.9 auf Seite 114).

Aus diesem Grund wird in [de:] ein Ansatz gewählt, der es zunächst ermöglicht, Formatierungen im Zusammenhang mit den zu ihrer Designaufgabe gehörigen anderen Formatierungen zu erstellen. Das Paradigma, das diesen Ansatz ermöglicht, heißt *Design by Example*. Es bringt, wie sich zeigen wird, neben dieser aufgabenorientierten Erstellung von Formatierungen aber noch weitere Vorteile mit sich.

Designspezifikation an Beispieldokumenten

Das Paradigma *Design by Example* gibt einen Ansatz vor, gemäß dem der Designspezifikationsprozess mit Hilfe von *repräsentativen*, logisch ausgezeichneten Beispieldokumenten erfolgt. Mit ihrer Hilfe werden Designspezifikationen erstellt, die auf beliebige ähnliche[5] Dokumente angewendet werden können.

Das Attribut repräsentativ für ein logisch ausgezeichnetes Dokument bedeutet dabei, dass es die zu verarbeitenden logischen Objekte in allen strukturellen Situationen beinhaltet, für die eine besondere Formatierung aufgestellt werden muss. Ein typisches Beispiel für diese strukturellen Situationen geben normale Paragraphen in einem Dokument ab: Oft wird hier zwischen dem ersten Paragraphen, der unmittelbar einer Überschift folgt und allen anderen Paragraphen unterschieden; der erste Paragraph erhält dann in seiner Formatierung als einziger keinen Erstzeileneinzug und benötigt deshalb eine individuelle Formatierung.

Konkret bedeutet das Paradigma *Design by Example* für [de:], dass die Baumdarstellung (siehe Abbildung 2.3) eines repräsentativen, logisch ausgezeichneten Dokumentes der Ausgangspunkt für die Entwicklung einer [em:] Designspezifikation ist. In diesem Dokument wird

[5] Z.B. alle Dokumente, die mit Hilfe der gleichen DTD erstellt worden sind (siehe 2.3.1).

dann jeweils durch die Erstellung *einer* Formatierung für *ein* spezifisches logisches Objekt gemäß dem schon in 6.2.3 erklärten Vorgehen von [de:] *eine* Designregel aufgestellt, die beschreibt, wie alle *gleichartigen* logischen Objekte, die im restlichen Dokument vorkommen, verarbeitet werden.

Die Gleichartigkeit von logischen Objekten ergibt sich dabei in [de:] über Umgebungsprädikate (siehe Definition 45 auf Seite 111), die für Designregeln aufgestellt werden können (siehe 6.2.11). Es besteht daher in [de:] die Möglichkeit, für ein logisches Objekt mehrere Designregeln zu definieren, wobei das Umgebungsprädikat entscheidet, welche Designregel angewendet wird. Eine Designregel, für die kein spezielles Umgebungsprädikat angegeben wird, ist eine *Default-Designregel*. Sie wird immer dann angewendet, wenn es für ein logisches Objekt keine passende Designregel mit einem Umgebungsprädikat gibt.

Somit ergibt sich insgesamt, dass in [de:] nicht für jedes einzelne in einem Dokument vorkommende logische Objekt individuell eine Formatierung aufgestellt werden muss, um die Verarbeitung eines Dokumentes zu spezifizieren. Statt dessen werden Designregeln aufgestellt, mit deren Hilfe den logischen Objekten eines Dokumentes, geknüpft an die Erfüllung eines Umgebungsprädikates, automatisch Formatierungen zugewiesen werden können.

6.2.6 Der Spezifikationsgraph

Die graphische Darstellung, die zur Aufstellung der Designregeln gemäß dem Paradigma *Design by Example* bearbeitet wird, heißt Spezifikationsgraph. Sie ist das zentrale Element in [de:] zur Visualisierung und ganzheitlichen Bearbeitung einer Designspezifikation.

Definition 58 (Spezifikationsgraph) *Der Spezifikationsgraph ist ein kombinierter Graph bestehend aus einem logischen Baum sowie den Formatierungen, die im Rahmen einer Designapplikation aufgrund der applizierten Designregeln für die einzelnen logischen Objekte des logischen Baumes angewendet werden.*

Bei der Erstellung einer neuen Designspezifikation mit Hilfe eines Spezifikationsgraphens beginnt man mit der *rohen* Darstellung eines logischen Baumes (siehe Abbildung 6.12 a). In dieser Darstellung können mit Hilfe von Drag and Drop Operationen (wie in Abschnitt 6.2.3 beschrieben) Formatierungen aufgebaut werden, die die Verarbeitung von logischen Objekten spezifizieren (siehe Abbildung 6.12 b).

Aufgrund eines in [de:] implementierten Mechanismus, der beim Anfügen des ersten Spezifikationsobjektes an ein logisches Objekt für die dadurch neu erstellte Formatierung auch zugleich eine Default-Designregel für das betroffene logische Objekt erstellt, wird diese Formatierung unverzüglich für alle anderen passenden logischen Objekte im Spezifikationsgraph angewendet. Der Spezifikationsgraph kann sich also bei jeder Drag and Drop Operation, die ein Designer auf ihm ausführt, nicht nur lokal, sondern an vielen Stellen verteilt gleichzeitig ändern (siehe Abbildung 6.13 und Abbildung 6.14).

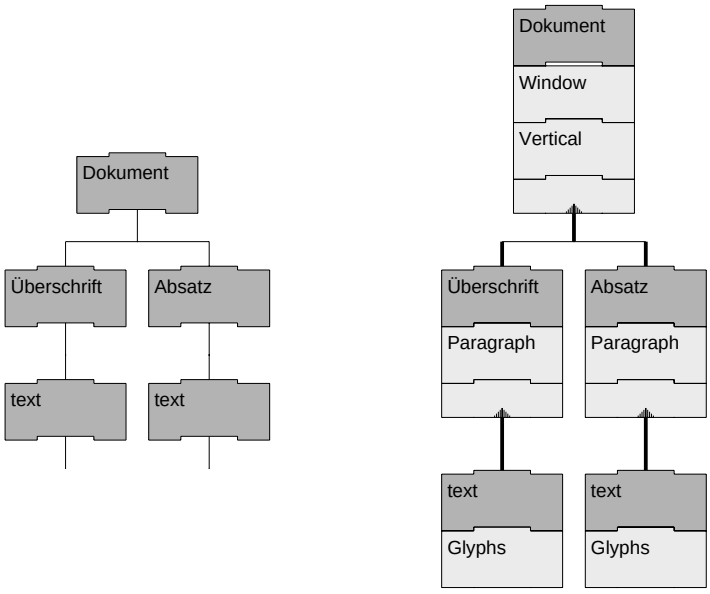

Abbildung 6.12: Der Spezifikationsgraph in zwei Phasen: Auf der linken Seite ist der Spezifikationsgraph in seiner einfachsten Form, einem logischen Baum zu sehen. Auf der rechten Seite der Abbildung ist er bereits vollständig mit einfachen Formatierungen versehen.

Die Konsequenz der Verwendung von Designregeln zur Anwendungssteuerung von Formatierungen ist also, dass ein und dieselbe Formatierung unter Umständen mehrfach im Spezifikationsgraph vorkommt. Dies wird in [de:] als *mehrfacher Darstellungszustand* für Formatierungen und die in ihnen vorkommenden Spezifikationsobjekte (siehe Abbildung 6.13) bezeichnet. Die in den Formatierungen vorkommenden logischen Objekte hingegen sind immer individuelle Instanzen, die aus dem logischen Baum herrühren. Bei der Bearbeitung eines der mehrfachen Vorkommen einer Formatierung im Spezifikationsgraph wird somit nicht eine individuelle Instanz einer Formatierung für ein logisches Objekt bearbeitet, sondern die Formatierung, die zu einer Designregel gehört. Aus diesem Grund wird nach der Änderung dieser Formatierung diese nun geänderte Formatierung im Rahmen einer Designapplikation auf alle logischen Objekte, die von der zugehörigen Designregel abgedeckt werden, angewendet und unverzüglich entsprechend im Spezifikationsgraph angezeigt.

Der mehrfache Darstellungszustand für Spezifikationsobjekte macht sich auch in der Farbgebung bemerkbar, die zur Anzeige des Zustands von Spezifikationsobjekten verwendet wird (siehe hierzu auch die Übersicht auf Seite 187). Zu den bereits eingeführten vier Zuständen normal, vorselektiert, selektiert und vorselektiert/selektiert gesellen sich noch fünf weitere Zustände für die Spezifikationsobjekte hinzu: Sie werden für die Spezifikationsobjekte verwendet, die im Rahmen einer Interaktion des Nutzers mit dem System *direkt* bearbeitet werden (siehe auch Abbildung 6.14). Unter direkter Verarbeitung versteht man in [de:] die unmittelbare Bearbeitung eines Objektes mit der Maus, während eine indirekte Bearbeitung sich über die mehrfache Darstellung eines Objektes ergibt; indirekt bearbeitet werden somit all diejenigen Spezifikationsobjekte, die in einem mehrfachen Darstellungszustand ein Spezifikationsobjekt repräsentieren, das in einer anderen Darstellung gerade direkt bearbeitet wird. Die folgende

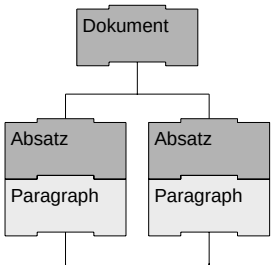

Abbildung 6.13: Einfacher Spezifikationsgraph mit einem mehrfachen Darstellungszustand für die Designregel, die einem logischen Objekt `Absatz` das Layoutobjekt `Paragraph` zuordnet. Beide Darstellungen des Spezifikationsobjektes `Paragraph` Verweisen auf das selbe Spezifikationsobjekt in einer Formatierung einer Designregel. Beide angezeigten Spezifikationsobjekte werden aufgrund *eines* Drag and Drop Vorganges, der auf einem beliebigen der beiden logischen `Absatz` Objekte geendet hat, dargestellt.

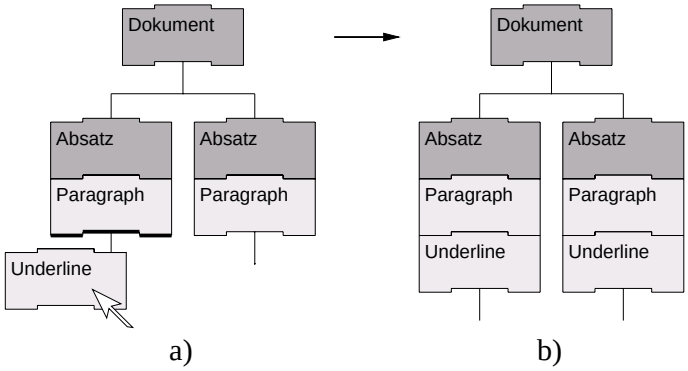

Abbildung 6.14: Änderung einer Formatierung im Spezifikationsgraph: Nach Änderung einer Formatierung in a, die zu einer Default-Designregel gehört, werden alle Vorkommen der Designregel im Spezifikationsgraphen geändert (siehe b). Das linke `Paragraph` Layoutobjekt in a wird direkt bearbeitet, das rechte indirekt.

Beschreibung der neuen Zustände der Spezifikationsobjekte bezieht sich bis auf den letzten Fall immer nur auf die Objekte, die direkt bearbeitet werden, für die indirekt bearbeiteten Spezifikationsobjekte gelten die bereits eingeführten Zustände.

- **Haupt-vorselektierter Zustand:**
  Für die direkt bearbeiteten Spezifikationsobjekte identisch zum Zustand vorselektiert.

- **Haupt-selektierter Zustand:**
  Für die direkt bearbeiteten Spezifikationsobjekte identisch zum Zustand selektiert.

- **Haupt-vorselektierter/haupt-selektierter Zustand:**
  Für die direkt bearbeiteten Spezifikationsobjekte identisch zum Zustand vorselektiert/selektiert.

- **Haupt-vorselektierter/selektierter Zustand:**
  Dieses Spezifikationsobjekt wird momentan direkt bearbeitet, ist aber nicht das Objekt das direkt haupt-selektiert wurde; es ist lediglich durch die Haupt-selektierung eines Objektes mit Hilfe einer Designregel und ihrer mehrfachen Darstellung mitselektiert worden.

- **Vorselektierter/Haupt-selektierter Zustand:**
  Dieses Spezifikationsobjekt ist zunächst haupt-selektiert worden und gelangt nun durch die Vorselektierung eines objektidentischen Mehrfachvorkommens von ihm in den Zustand Vorselektiert/Haupt-selektiert.

Die große Anzahl an möglichen Zuständen für ein Spezifikationsobjekt mag auf den ersten Blick verwirrend und unzugänglich für die Nutzer erscheinen. Diese Zustände ergeben sich aber aus natürlichen Gegebenheiten, die im Rahmen der interaktiven Arbeit eines Designers mit einem Spezifikationsgraphen entstehen (siehe Abbildung 6.15). Beobachtungen haben gezeigt, dass bei geeigneter Farbwahl für die Zustände sie die Nutzer bei der Arbeit mit dem System auf effiziente Weise unterstützen.

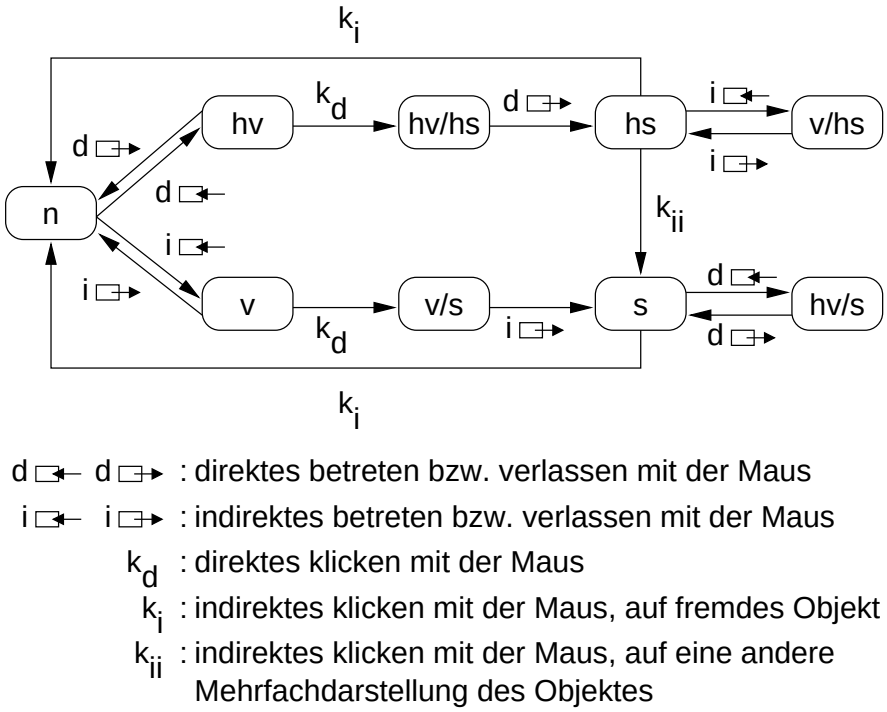

Abbildung 6.15: Zustandsübergänge eines Spezifikationsobjektes: Obiges Zustandsübergangsdiagramm zeigt die neun möglichen Zustände eines Spezifikationsobjektes auf. `n` steht dabei für normal, `v` jeweils für vorselektiert, `s` für selektiert und `h` für die Eigenschaft, Hauptobjekt bezüglich einer Eigenschaft zu sein.

Die Pipeline-Metapher

Ein Spezifikationsgraph beinhaltet bei Beendigung einer Designspezifikation für die meisten (wenn nicht alle) der in ihm vorkommenden logischen Objekte eine Formatierung. Diese Formatierungen beschreiben in ihrer Gesamtheit statisch die Anweisung (Designspezifikation), wie ein Dokument zu verarbeiten ist. Der Spezifikationsgraph bringt nun durch seine Beschaffenheit die Möglichkeit mit sich, dass man diese Verarbeitungsanweisung direkt in ein leicht verständliches Prozessmodell übersetzen kann, das dynamisch anzeigt, in welcher Art und Weise das Dokument verarbeitet wird: Ein Pipelinesystem.

Entfernt man gedanklich aus einem Spezifikationsgraphen alle logischen Objekte, so ergibt sich eine Hierarchie von Spezifikationsobjekten, die man als Pipelinesystem betrachten kann. In diesem Pipelinesystem wird konzeptionell von den Quellobjekten Inhalt in Form eines Flusses ausgesendet, dieser von den transienten Objekten verarbeitet und weitergegeben und schließlich von den Zielobjekten empfangen und entsprechend dem Zielobjekt umgesetzt (siehe Abbildung 6.16). In der Designspezifikationsmethodik [em:] wird ein Pipelinesystem auch Designstruktur genannt.

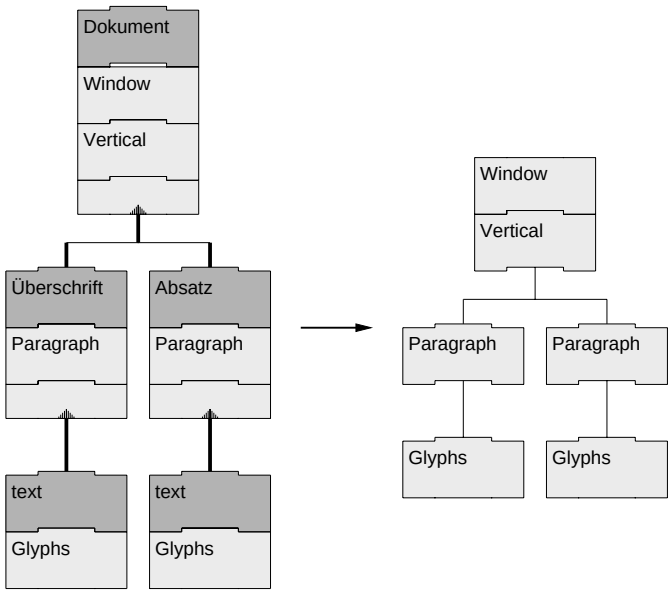

Abbildung 6.16: Übergang eines Spezifikationsgraphen zu einem Pipelinesystem

Um die Mächtigkeit einer Designspezifikation zu erweitern, werden in [de:] weiterhin Konnektoren eingeführt. Dies sind Spezifikationsobjekte, die zur bisher nicht beschriebenen Spezifizierung verwendet werden, an welchen Stellen die Pipeline vom unteren Ende einer Formatierung aus weiter nach unten gebaut werden soll[6]. Diese Einführung der Konnektoren bringt erstmals die Möglichkeit mit sich, dass im Verlauf einer Designspezifikation aus dem Spezifikationsgraphen auch tatsächlich ein Graph wird. Werden auf beliebige Weise in einer Forma-

[6]Das erste an ein logisches Objekt angefügte Spezifikationsobjekt bewirkt in [de:] automatisch die Anfügung eines Konnektors.

tierung zwei Konnektoren verwendet, führen beide Verbindungslinien zum darunterliegenden logischen Objekt und werden dort vereinigt (siehe Abbildung 6.17).

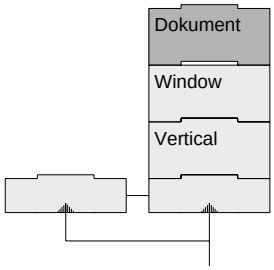

Abbildung 6.17: Formatierung mit zwei Konnektoren: Von der dargestellten Formatierung gehen, verursacht durch zwei Konnektoren, zwei Pipelinerohre nach unten ab. Sie bewirken, dass die weiter unten folgenden Formatierungen der logischen Objekte doppelt verarbeitet werden.

Diese Zusammenführung der Verbindungslinien zu einem logischen Objekt bringt mit sich, dass das durch die Designspezifikation berechnete Pipelinesystem strukturell nicht mehr mit dem Spezifikationsgraph übereinstimmt. Die Formatierung des doppelt bezogenen logischen Objektes wird zweifach in dem Pipelinesystem vorkommen. Die Information über ein derartiges mehrfaches Vorkommen einer Formatierung wird den Nutzern im Spezifikationsgraphen mit Hilfe einer Zahl[7], die rechts oben neben der Formatierung angezeigt wird, zugänglich gemacht (siehe Abbildung 6.18). Werden innerhalb einer Formatierung ab einer bestimmtem Stelle einer Formatierung die vorkommenden Spezifikationsobjekte unterschiedlich oft ausgewertet (z.B. aufgrund eines `Repeater` Spezifikationsobjekt, siehe Seite 126) wird dies ebenfalls durch eine Nummerierung neben den betroffenen Spezifikationsobjekten in der Formatierung angezeigt.

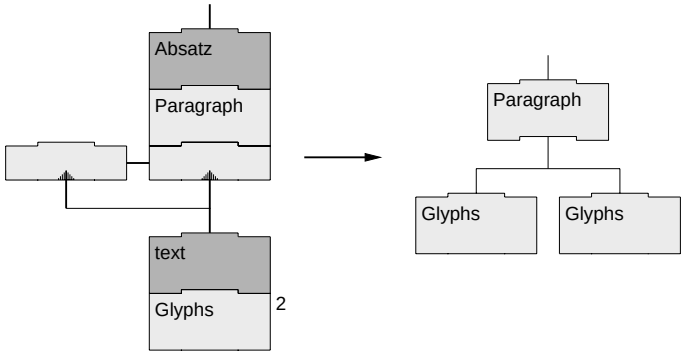

Abbildung 6.18: Mehrfachverwendung von Formatierungen in der Designstruktur: Aufgrund der Verwendung von zwei Konnektoren in der oberen Formatierung für das logische `Absatz` Objekt wird die untere Formatierung für das logische `text` Objekt zweimal in der Designstruktur verwendet. Dies wird im Spezifikationsgraphen durch die `2` rechts oben neben der entsprechenden Formatierung angezeigt.

[7] Nur die Zahlen 1 bis 9 werden direkt angezeigt, bei größeren Zahlen wird nur noch das Zeichen * angezeigt.

Der Zustand des Flusses, der sich in einem Pipelinesystem ergeben kann, wird im Spezifikationsgraph in [de:] graphisch visualisiert. Zu jedem Zeitpunkt kann ein Fluss dabei in einem von drei möglichen Zuständen sein (siehe hierzu auch Seite 100):

- **possible:**
  An dieser Stelle könnte ein Fluss in dem Pipelinesystem fließen, wenn von weiter unten etwas ankommen würde. Dies wird durch eine dünne Verbindungslinie angezeigt.
- **available:**
  An dieser Stelle ist im Fluss Inhalt vorhanden, weiter oben kommt jedoch kein Zielobjekt vor, das diesen Fluss akzeptieren kann. Diese Eigenschaft wird durch eine dicke Verbindungslinie repräsentiert.
- **used:**
  An dieser Stelle fließt der Fluss. Die Verbindungslinie wird dick-gestrichelt und bei Wunsch auch animiert im Spezifikationsgraph dargestellt.

Wertsetzung für Spezifikationsattribute

Spezifikationsobjekte haben jeweils einen Satz von Spezifikationsattributen. Die Werte für diese Attribute können von den Designern so gesetzt werden, dass die durch ein Spezifikationsobjekt realisierte Funktionalität bei Beachtung der Attributierung exakt die Verarbeitung spezifiziert, die die Designer wünschen. Die zur Verfügung stehenden Spezifikationsattribute sind dabei von den Sorten `Bool`, `Number`, `Length` oder `String` (siehe 4.9.3).

Da die Spezifikationsobjekte im Rahmen einer Designapplikation nicht individuell durch die Nutzer an logische Objekte zugewiesen werden, sondern mit Hilfe von Formatierungen aufgrund von Designregeln in Form eines automatischen Prozesses, ist es bisher in [de:] für die Nutzer außer unter Rückgriff auf kontextabhängige Designregeln nicht möglich, gezielt an ausgesuchten Stellen gewisse Parameter für diese Spezifikationsobjekte mit ausgewählten Werten zu besetzen. Dieser Rückgriff auf kontextabhängige Designregeln ist jedoch nicht immer der optimale Weg. Ein Beispiel, das diese Problemstellung verdeutlicht, ist der Text innerhalb von Hervorhebungen: Er ist oft in kursiver Schrift gesetzt, Hervorhebungen in Hervorhebungen sind jedoch wieder in normaler Schrift, u.s.w.. Beachtet man, dass diese Beeinflussung des Schriftmerkmals kursiv noch von weiteren Faktoren anhängen kann (z.B. wird auch oft innerhalb von Definitionen auf kursive Schrift gewechselt, was wiederum Auswirkungen auf die Hervorhebungen hat), würden in diesen Fällen unverhältnismäßig viele kontextabhängige Designregeln nötig werden, deren Formatierung bis auf die Attributierung identisch wäre. Bei einer Änderung des Designs müssten sie alle konsistent geändert werden. Aus diesem Grund ist es nötig, ein zu den kontextabhängigen Designregeln alternatives Verfahren zur Verfügung zu haben, mit dessen Hilfe in derartigen Situationen die Spezifikationsattribute gezielt gesetzt werden können.

Dieses Verfahren beruht in [de:] auf dem sogenannten Spezifikationsvariablenfluss. Für jedes Spezifikationsattribut, das für ein beliebiges Spezifikationsobjekt in [de:] existiert, gibt es eine entsprechende Spezifikationsvariable gleichen Namens und gleicher Sorte, die mit einem

sinnvollen Wert vorbelegt ist. Vom Konzept her werden die Werte dieser Spezifikationsvariablen an der Wurzel des Spezifikationsgraphen, aus dem sich das Pipelinesystem ergibt, zur Verfügung gestellt und werden von hier aus nach unten weitergegeben. Auf diesem Weg stehen die Werte dieser Spezifikationsvariablen zur Wertsetzung der Spezifikationsattribute der Spezifikationsobjekte zur Verfügung: In den Ausdrücken (siehe 4.9), die zur Wertsetzung von Spezifikationsattributen verwendet werden, können die Spezifikationsvariablen verwendet werden.

Das `Inheritance` Auswertungsobjekt
Zur Manipulation der Werte von Spezifikationsvariablen im Verlauf einer Auswertung von oben nach unten, wie sie z.B. bei der Aufgabenstellung *Hervorhebung in Hervorhebung* nötig wird, kann das `Inheritance` Auswertungsobjekt verwendet werden. Mit seiner Hilfe kann der Wert für eine Spezifikationsvariable ab einer bestimmten Stelle im Spezifikationsgraph nach unten hin verändert werden (siehe Abbildung 6.19). Zur Steuerung dieser Funktionalität verfügt das `Inheritance` Auswertungsobjekt über eine frei erweiterbare Liste von Spezifikationsvariablen, die es manipulieren kann.

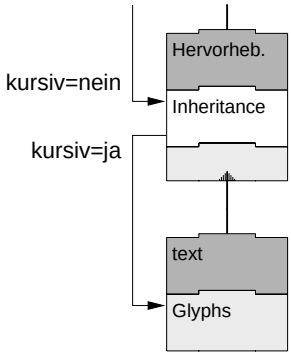

Abbildung 6.19: Verwendung eines `Inheritance` Auswertungsobjektes: Mit Hilfe des in dem Spezifikationsgraphen verwendeten `Inheritance` Auswertungsobjektes wird in einer Hervorhebung die zu verwendende Schrift auf kursiv gesetzt.

Neben den automatisch im System vorgegebenen Spezifikationsvariablen können die Nutzer in [de] noch eigene Spezifikationsvariablen einführen. Diese werden unter anderem auch zur Realisierung der aktiven Berechnung von Werten im Rahmen einer Designapplikation verwendet. Als typisches Beispiel ist hier die Nummerierung von Überschriften zu nennen, die in logisch ausgezeichneten Dokumenten nicht immer automatisch enthalten sein muss.

Das `Modifier` Auswertungsobjekt
Da Nummerierungen nicht nur hierarchisch von oben nach unten zur Verfügung stehen müssen, sondern noch dazu auch in der Regel sequentiell von links nach rechts beim Durchlaufen eines Spezifikationsgraphens verändert werden, muss diese Funktionalität ebenfalls mit einem Mechanismus nachgebildet werden. Dies geschieht in [de:] mit Hilfe der `Modifier` Auswertungsobjekte, die eine Spezifikationsvariable entsprechend dieser Aufgabenstellung manipulieren können (siehe hierzu Abbildung 6.20). Im Gegensatz zum `Inheritance` Auswertungsobjekt verändert ein `Modifier` Auswertungsobjekt den Wert einer Spezifikationsvariablen erst nach Abarbeitung seines Unterbaumes für den Rest des Spezifikationsgraphens. Die genaue

Funktionsweise dieses Mechanismus der Spezifikationsvariablen und ihrer Wertberechnung ist in Abschnitt 4.4.4 dieser Arbeit beschrieben.

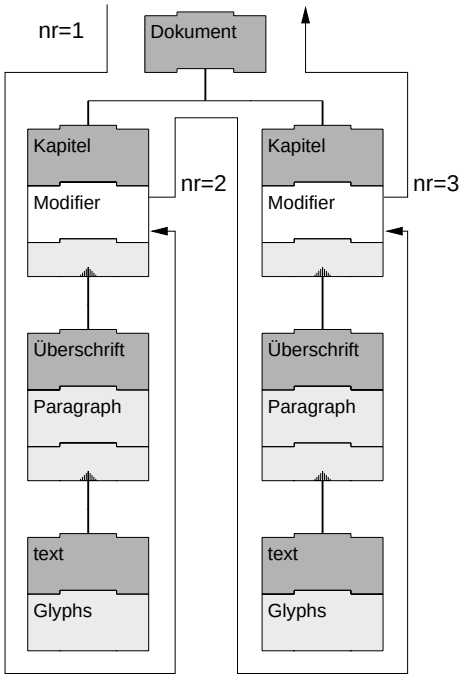

Abbildung 6.20: Verwendung eines `Modifier` Auswertungsobjektes: Mit Hilfe der im Spezifikationsgraphen vorkommenden `Modifier` Auswertungsobjekte kann die Nummerierung (mit der Spezifikationsvariablen `nr`) für die Kapitel vollzogen werden.

Der Zugriff auf die logischen Attribute eines logisch ausgezeichneten Dokumentes erfolgt in [de:] analog zu dem Verfahren der Weitergabe der Werte von Spezifikationsvariablen im Spezifikationsgraph. Die Werte eines logischen Attributes stehen ab der Stelle in einem Spezifikationsgraph nach unten zur Verfügung, an der im Spezifikationsgraph das zugehörige logische Objekt steht. Ein typischer Zugriff auf den Wert eines logischen Attributes findet in den `Glyphs` Spezifikationsobjekten statt, um den Text, den sie realisieren sollen, zu erhalten: Logische `text` Objekte stellen in dem logischen Attribut `@input` den Text zur Verfügung, den sie repräsentieren (siehe Abbildung 6.21). Näheres zu den logischen Attributen siehe in Abschnitt 4.3.

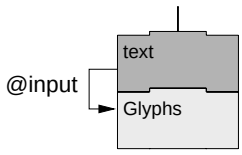

Abbildung 6.21: Verwendung von logischen Attributen: Von dem in der Abbildung vorkommenden `Gylphs` Layoutobjekt wird das logische Attribut `@input` des logischen Objektes `text` verwendet.

Versiegte Quellobjekte

Der Spezifikationsvariablenfluss und die Werte der logischen Objekte können sich in [de:] auch auf den Zustand und somit die gewählte Füllfarbe für Quellobjekten auswirken. Wie *echte* Quellen symbolisieren auch die Quelleobjekte in [de:], die in einem Spezifikationsgraph verwendet werden, den Ursprung von einem Fluss. Dieser Fluss kann, wie auch in der Realität, in [de:] versiegen. Dies ist z.B. dann der Fall, wenn ein `Glyphs` Spezifikationsobjekt keine Buchstaben realisiert (z.B. weil es die Verarbeitung eines Dokumentteiles spezifiziert, für den noch kein Text vorgegeben ist). Dieses `Glyphs` Layoutobjekt wird dann in einem Spezifikationsgraph anders als ein `Glyphs` Layoutobjekt dargestellt, das Buchstaben realisiert. Abbildung 6.22 zeigt beide möglichen Darstellungen in einer Gegenüberstellung.

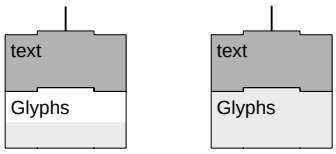

Abbildung 6.22: Die zwei Zustände eines Quellobjektes: Links ist das `Glyphs` Layoutobjekt in einem *versiegten* Zustand dargestellt, rechts in einem *quellenden* Zustand.

Erweiterung des Spezifikationsgraphen - Views

Ein gegebenes logisch ausgezeichnetes Dokument wird in der Regel nicht nur unter einem Aspekt verarbeitet, sondern unter mehreren. So soll in der Regel aufgrund eines gegebenen Dokumentes ein Inhaltsverzeichnis und der normale Textteil erstellt werden. Diese verschiedenen Sichten, die bei der Betrachtung eines Dokumentes unter verschiedenen Aspekten entstehen, werden in [de:] *Views*[8] genannt.

Aufgrund dieser Erkenntnis der Praxisrelevanz von Views ist es nötig, eine derartige Mehrfachverarbeitung eines Dokumentes auch in [de:] zu ermöglichen. Aus diesem Grund wird in [de:] das Konzept der bisherigen *einfachen* Designregeln auf Designregeln erweitert, die nur für bestimmte Views gültig sind. Somit kann man z.B. für einen normalen Paragraphen eines Dokumentes eine Designregel für den View *Inhaltsverzeichnis* erstellen, deren Formatierung leer ist (ein normaler Paragraph trägt keinen Inhalt zum Inhaltsverzeichnis bei) und eine Designregel für den View *Textteil*, die als Formatierung wie üblich ein `Paragraph` Layoutobjekt enthält. Ein analoges Vorgehen ist für die logischen `text` Objekte erforderlich.

Es ist notwendig, die verschiedenen Views auf ein Dokument mit den dazugehörigen Designregeln in [de:] darstellen und bearbeiten zu können. Hierdurch ergibt sich jedoch noch nicht die Notwendigkeit, das Konzept des Spezifikationsgraphen zu ändern: Die einzelnen isolierten Views sind konzeptionell jeweils identisch zu dem bisherigen einzigen View, der in [de:] verfügbar ist.

Oft ist es jedoch auch nötig, die verschiedenen Views auf ein Dokument *kombiniert* verwenden zu können. In diesem Fall müssen zur Erstellung geeigneter Designspezifikationen zum einen im Spezifikationsgraph mehrere Views parallel auf einmal dargestellt werden können.

[8] Views werden in [de:] mit Nummern bezeichnet, im folgenden Text jedoch oft mit Namen benannt.

Zum anderen muss es aber auch ein Konstrukt in [de:] geben, mit dessen Hilfe verschiedene Views in einem neuen View kombiniert und somit beispielsweise in einem `Window` Layoutobjekt zur späteren Darstellung zusammengeführt werden können.

Die gemeinsame Anzeige mehrerer Views geschieht in [de:] in einem erweiterten Spezifikationsgraphen, der zwar komplexer als ein normaler Spezifikationsgraph ist, aufgrund seines systematischen Aufbaus aber ebenso leicht zu verstehen ist.

Definition 59 (Erweiterter Spezifikationsgraph) *Der erweiterte Spezifikationsgraph ist ein kombinierter Graph bestehend aus einem logischen Baum sowie den Formatierungen, die im Rahmen einer Designapplikation aufgrund der applizierten Designregeln für die einzelnen logischen Objekte des logischen Baumes angewendet werden. Die einzelnen Formatierungen der Designregeln für die logischen Objekte sind dabei gemäß ihrer aufsteigenden View-Nummerierung mit Hilfe von Verbindungslinien als Kinder der logischen Objekte angeordnet.*

Ein Beispiel für einen erweiterten Spezifikationsgraphen ist in Abbildung 6.23 zu sehen. Er stellt einen Ausschnitt auf eine Designspezifikation dar, in der zwei Views für ein Dokument bearbeitet werden: Einer für das Inhaltsverzeichnis und einer für den normalen Textteil. Die einzelnen Views werden dabei in der Darstellung farbig kodiert. Jeder View und somit die Linien, die die Formatierungen der einzelnen Views mit den logischen Objekten verbinden, erhält eine eigene Farbe, die in einer Konfigurationsdatei frei vorgebbar ist.

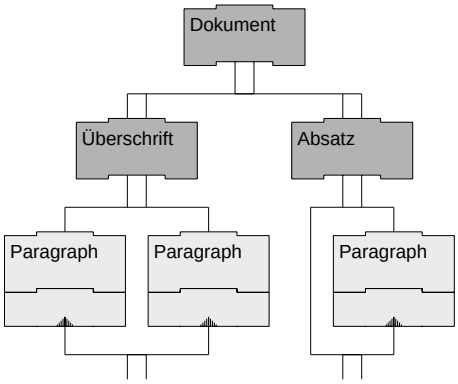

Abbildung 6.23: Erweiterter Spezifikationsgraph: In obiger Darstellung sind zwei Views aktiviert. Der linke View unter den logischen Objekten ist jeweils für das Inhaltsverzeichnis, der rechte für den normalen Textteil. Da ein normaler Paragraph nichts zu einem Inhaltsverzeichnis beiträgt, ist die entsprechende linke Formatierung für ihn leer.

Das Konstrukt, mit dessen Hilfe in [de:] verschiedene Views zusammengeführt und gemeinsam verwendet werden können, beruht auf einer Erweiterung der Konnektoren: Für sie kann angegeben werden, von welchem View Formatierungen von unten zum Aufbau des Pipelinesystems für sie verwendet werden sollen. Abbildung 6.24 zeigt hierfür ein Beispiel auf, das auf Abbildung 6.23 aufbaut. Dort wird einem `Window` Layoutobjekt sowohl das Inhaltsverzeichnis als auch der normale Textteil eines Dokumentes verwendet. In Abbildung 6.25 wird das zu dem in Abbildung 6.24 dargestellten erweiterten Spezifikationsgraphen gehörige Pipelinesystem dargestellt.

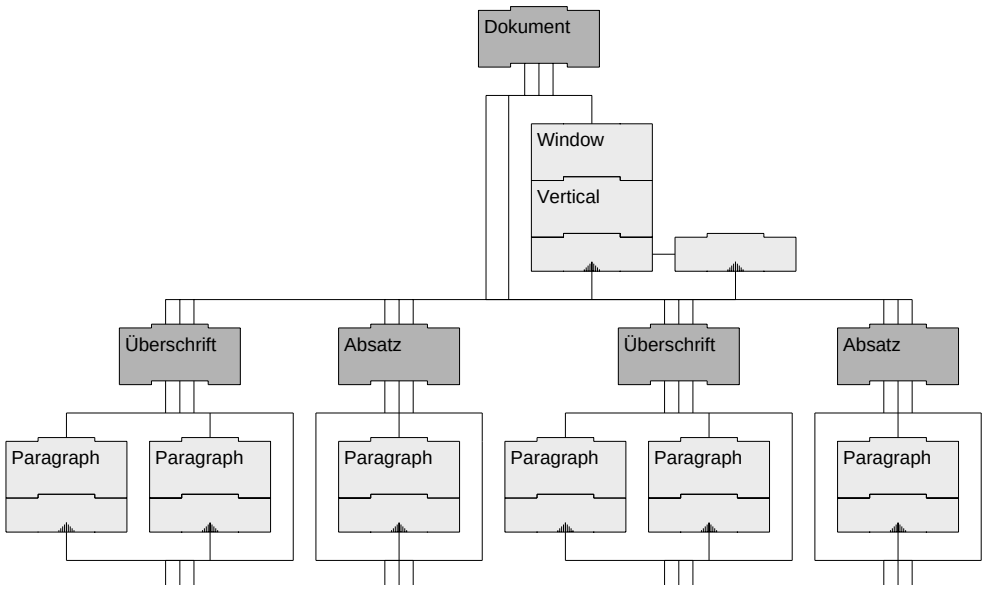

Abbildung 6.24: Konnektoren im erweiterten Spezifikationsgraph: Die in obiger Darstellung eines Spezifikationsgraphen mit drei Views platzierten Konnektoren bewirken die Verwendung von zwei unterschiedlichen Views in der Layoutobjektkombination `Window/Vertical`. Dies realisiert die Erstellung einer Präsentation für das Dokument, die sowohl ein Inhaltsverzeichnis als auch den normalen Textteil beinhaltet.

Anreicherung der logischen Struktur - das `SetLogAtt` Spezifikationsobjekt

Stellt man sich vor, dass aufgrund einer fehlenden Nummerierung im gegebenen logisch ausgezeichneten Dokument die Überschriftennummerierung mit Hilfe der in [de:] gegebenen Möglichkeiten realisiert werden muss, so ist dies durch die Verwendung von `Modifier` Auswertungsobjekten erreichbar (siehe Abbildung 6.20 auf Seite 205). Verwendet man nun mehrere Views zur Realisierung des Designs des Dokumentes, so ist mit den bisherigen Möglichkeiten in jedem View erneut eine eigene Nummerierung zu realisieren – eine Notwendigkeit, die Redundanz und die damit verbundenen Nachteile in eine Designspezifikation einführt.

Aus diesem Grund wird mit Hilfe der `SetLogAtt` Auswertungsobjekte in [de:] eine Möglichkeit eingeführt, mit deren Hilfe für Spezifikationsvariablen berechnete Werte in logischen Objekten als *Aufhängepunkte* abgelegt werden können. Es findet eine Anreicherung des logischen Baumes um logische Attribute statt, die dann jederzeit wie die von vornherein im logischen Baum vorhandenen logischen Attribute verwendet werden können.

Ein `SetLogAtt` Auswertungsobjekt hat wie auch schon die Auswertungsobjekte `Inheritance` und `Modifier` eine benutzerdefinierbare Liste von Spezifikationsvariablen, die es verwaltet. Die in dieser Liste enthaltenen Spezifikationsvariablen werden nach der Berechnung eines durch die Nutzer vorgebbaren Ausdruckes (z.B. `nr:=nr+1`) in dem logischen Objekt abgelegt, für das die Formatierung verwendet wird (siehe Abbildung 6.26).

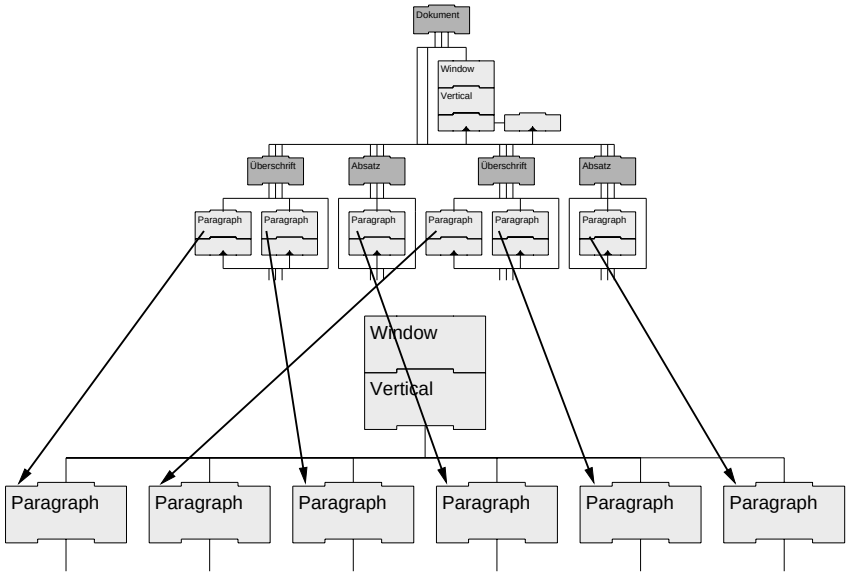

Abbildung 6.25: Pipelinesystem zu Abbildung 6.24: Die Pfeile in obiger Abbildung zeigen den Anbau von Formatierungen aufgrund der in der oberen Formatierung für das logische `D` Objekt vorkommenden Konnektoren auf.

Vorteile des Spezifikationsgraphen

Versucht man an dieser Stelle kurz die Vorteile des Spezifikationsgraphen bei der Erstellung von Designregeln herauszuarbeiten, so ist klar, dass dies die Vorteile sind, die sich gegenüber der Erstellung in textueller Form oder im Formatierungs-Fenster ergeben. In beiden letztgenannten Fällen findet die Erstellung der einzelnen Designregeln in isolierter Form statt: Jede Designregel wird für sich und nur in losem Zusammenhang mit den anderen Designregeln erarbeitet.

Im Spezifikationsgraph hingegen können die Designregeln für Teilaufgaben einer gesamten Designspezifikation aufgabenorientiert zusammenhängend erstellt werden. Es werden sogar nicht nur einzelne Designregeln von den Designern erarbeitet, sondern eine ganzheitliche Verarbeitungsanweisung, die eine gesamte Designaufgabe (deren Bearbeitung mehrere Designregeln erfordert) übernimmt. Dieser Punkt kommt besonders dann zum Tragen, wenn es sich bei den logischen Objekten, die im Rahmen eines Teilbereichs einer Designspezifikation bearbeitet werden, um räumlich eng zusammenliegende logische Objekte handelt. Ein geeignetes Beispiel hierfür ist z.B. die Erstellung des Designs einer Liste (siehe hierzu Abbildung 6.27).

Aber auch bei *räumlich verteilten* Aufgabenstellungen wie z.B. der Einführung einer Nummerierung bietet der Spezifikationsgraph eine gute Arbeitsgrundlage, wie Abbildung 6.28 zeigt. In der Designspezifikation, die in dem Spezifikationsgraphen in Abbildung 6.28 dargestellt ist, werden die Abschnittsüberschriften eines Dokumentes, das aus beliebig tief verschachtelten Abschnitten besteht, automatisch durchnummeriert. Dazu werden zwei Spezifikationsvariablen eingeführt: `counter` zählt die Abschnitte der aktuellen Abschnittsebene durch, `numstring` enthält die Nummerierung für die jeweils aktuelle Abschnittsebene in Stringform. Die Formatierung eines jeden logischen `Abschnitt` Objektes enthält ein `Inheritance` Auswertungsobjekt, welches in `numstring` die aktuelle Nummerierung aus dem alten `numstring`

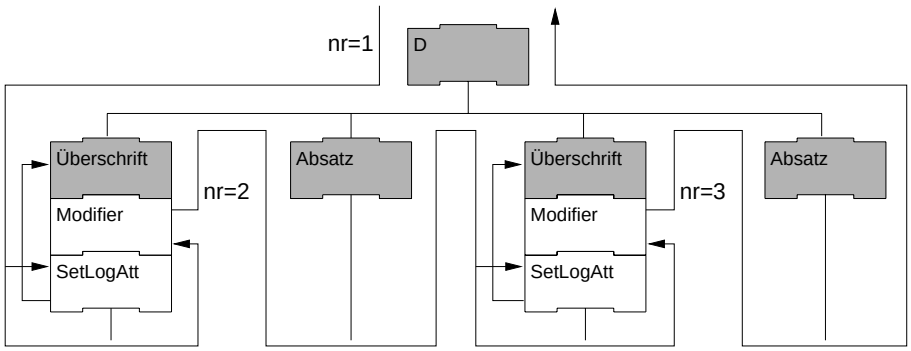

Abbildung 6.26: Anreicherung des logischen Baumes mit dem `SetLogAtt` Auswertungsobjekt: Die in obigem Spezifikationsgraph für das logische `Überschrift` Objekt verwendete Formatierung berechnet mit Hilfe des `Modifier` Auswertungsobjektes die Nummerierung für die Überschriften und reichert das darüberliegende logische Objekt jeweils mit dem Ergebnis der Berechnung an. Auf dieses Ergebnis kann später in weiteren Formatierungen jederzeit zugegriffen werden.

Wert sowie angehängter `counter` Spezifikationsvariable, gefolgt von einem Punkt, berechnet weitergibt. Zusätzlich wird dort der `counter` auf `1`, den Zählerstartwert für eine neue Abschnittsebene, zurückgesetzt. Das `Glyphs` Layoutobjekt, das den Text für die Überschrift spezifiziert, erhält den String `numstring` als Präfix vor dem eigentlichen Text der Überschrift. Durch das `Modifier` Auswertungsobjekt in der Formatierung der Abschnitte wird `counter` nach der Auswertung eines kompletten Abschnitts für die folgenden Abschnitte auf gleicher Ebene um eins erhöht.

Als unmittelbare Konsequenz seiner Form und den darin angeordneten Objekten ermöglicht der Spezifikationsgraph seinen Nutzern einen Überblick darüber zu erhalten, wie verschiedene Gruppen von Designregeln in einer Designspezifikation miteinander interagieren – der Spezifikationsgraph zeigt die Formatierungen nicht nur als *leblose* Objekte an, sondern durch die ihm zugrundeliegende Pipelinemetapher auch, wie sie *leben* und in welchem Zustand sie sich dabei befinden.

Weiterhin wird im Spezifikationsgraph mit Hilfe der verschiedenen Objektklassen und ihrer unterschiedlichen Visualisierung eine klare Trennung zwischen den verschiedenen Aufgaben, die in einem Spezifikationsgraph realisiert werden können, erzielt. Der wichtigste Punkt, der durch dieses systematische Vorgehen erreicht wird, ist die deutliche Unterscheidung zwischen der Berechnung von Attributwerten mit Hilfe der Spezifikationsvariablen, die in den Auswertungsobjekten manipuliert werden können und deren letztendliche Verwendung in den Layoutobjekten.

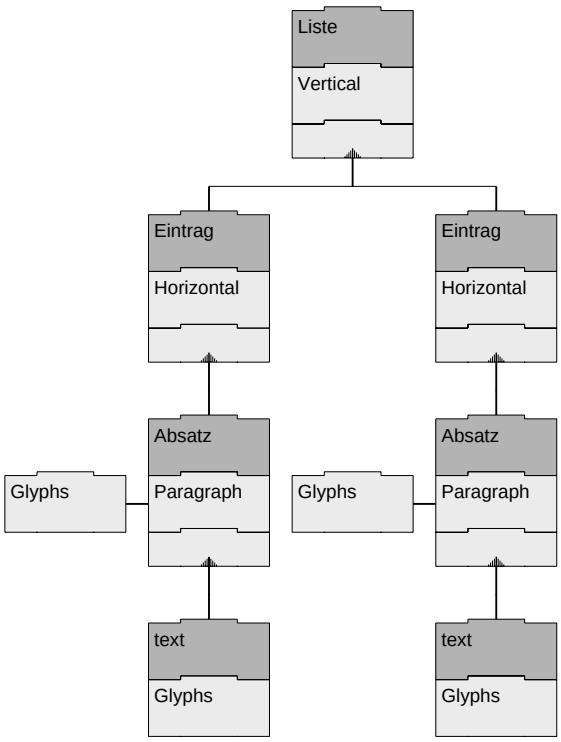

Abbildung 6.27: Spezifikationsgraph zur Erstellung des Designs einer Liste

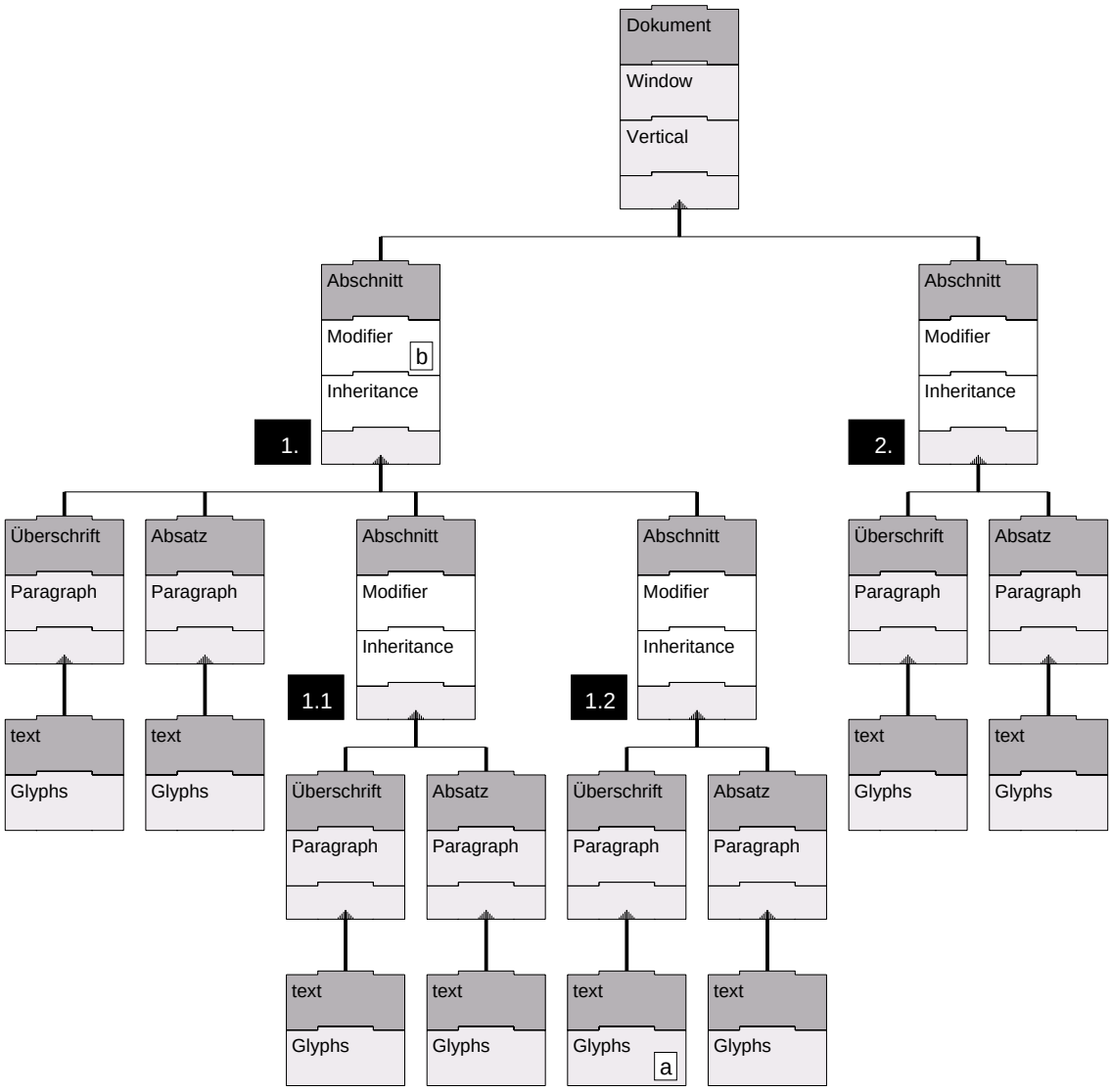

Abbildung 6.28: Spezifikationsgraph zur Nummerierung eines Dokumentes

6.2.7 Das Hauptfenster

Das Hauptfenster von [de:] (siehe Abbildung 6.29) ist wie auch schon das Formatierungs-Fenster horizontal in zwei Bereiche eingeteilt. Ebenso wie dort sind auch hier auf der linken Seite die zur Verfügung stehenden Spezifikationsobjekte in Gruppen unterteilt zur Auswahl durch die Nutzer angeordnet. Auf der rechten Seite des Hauptfensters ist der Spezifikationsgraph zu finden. Dieser kann mit Hilfe der in Abschnitt 6.2.3 beschriebenen Methoden der direkten Manipulation und den in Abschnitt 6.2.6 vorgestellten Konzepten zur Erstellung von Designregeln verwendet werden.

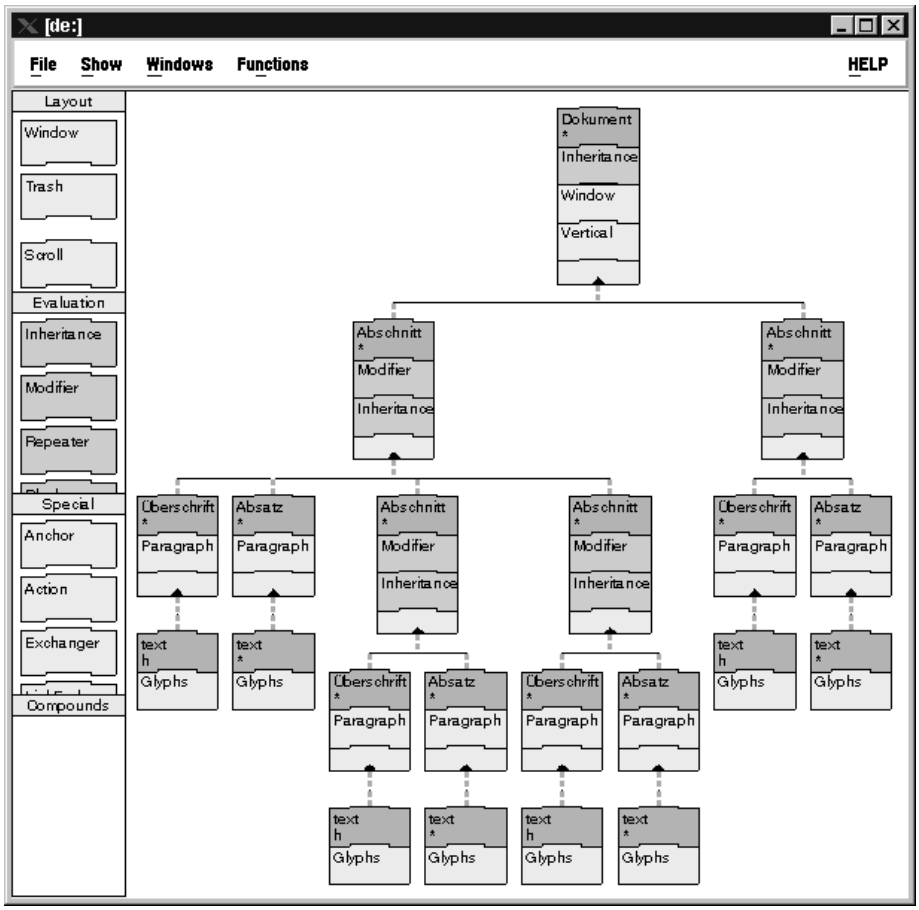

Abbildung 6.29: Das Hauptfenster von [de:]: Auf der rechten Seite ist der Spezifikationsgraph aus Abbildung 6.28 zu sehen.

Der Spezifikationsgraph

Wie im Formatierungs-Fenster ist es im Hauptfenster möglich, die in den einzelnen Bereichen enthaltenen Darstellungen zu manipulieren (siehe hierzu Auflistung auf Seite 194). Hinzu kommt beim Hauptfenster die Möglichkeit, dass Teilbäume des Spezifikationsgraphens animiert ein- und ausgeklappt werden können. Dies wird durch die Nutzer angestoßen, indem sie

mit der rechten Maustaste ein logisches Objekt im Spezifikationsgraphen anklicken. Zweck dieser Funktionalität ist es, den Nutzern die Möglichkeit zu geben, das ihnen zur Erstellung einer Designspezifikation an die Hand gegebene logisch ausgezeichnete Dokument so zur Bearbeitung am Bildschirm zur Verfügung zu haben, dass sie sich optimal auf ihre gegenwärtige Aufgabe im Rahmen der aktuell ausgeführten Designspezifikation konzentrieren können. Eine Möglichkeit hierzu ist eben die *Ausblendung* momentan nicht-relevanter Dokumentteile und der für sie schon erstellten Designregeln.

Für den Fall, dass ein erweiterter Spezifikationsgraph dargestellt wird, kann ein Spezifikationsobjekt nicht nur wie in der Erklärung des Einfügens von Spezifikationsobjekten auf Seite 191 als erstes Objekt in eine Formatierung vorgenommen werden, indem es in die Nähe des logischen Objektes gebracht wird. Statt dessen kann es auch auf einer der dargestellten Verbindungslinien für die Multi-View Darstellung fallengelassen werden (siehe Abbildung 6.30), um eine Designregel für den entsprechenden View zu erstellen.

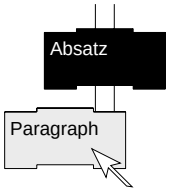

Abbildung 6.30: Drag and Drop Operation in einer Multi-View Darstellung: Ein Spezifikationsobjekt kann hier auf einer Verbindungslinie für einen speziellen View fallengelassen werden.

Wahlweise kann anstelle des Spezifikationsgraphen auch der *reine* Baum der logischen Objekte angezeigt werden. Die Umschaltung erfolgt über den Menüpunkt `Show - Only Logical Objects` bzw. `Show - Layout Objects`. Neben dieser vollkommenen Ausschaltung der Formatierungen kann der Spezifikationsgraph mit Hilfe eines Steuerfensters, das über den Menüpunkt `Windows - Views` aktivierbar ist, so manipuliert werden, dass er nur für bestimmte Views die Formatierungen anzeigt.

Um Platz zu sparen werden logische Objekte mit ihren Formatierungen so dicht wie möglich an ihrem logischen Vorgänger-Objekt gezeichnet. Dadurch werden im Allgemeinen zwei logische Objekte, die die gleiche logische Tiefe im Dokumentbaum haben, in unterschiedlicher Höhe gezeichnet. Optional können solche Objekte immer auf gleicher Höhe gezeichnet werden. Der Menüpunkt `Show - Separate Layers` bzw. `Show - Nonseparate Layers` dient zum Ein- und Ausschalten dieser Option.

Die Designstruktur

Über den Menüpunkt `Window - Design Structure` kann ein Fenster aktiviert werden, in dem die Designstruktur zum aktuell im Hauptfenster angezeigten Spezifikationsgraph dargestellt wird. In dieser Anzeige ist es über ein Menü möglich, sich neben den Spezifikationsobjekten noch die logischen Objekte anzeigen zu lassen, aufgrund derer die Spezifikationsobjekte in die Designstruktur aufgenommen wurden – ein Punkt, der sich bei größeren Designspezifikationen als sehr nützlich erwiesen hat. Als Beispiel hierzu siehe Abbildung 6.31, in der

diese angereicherte Variante der Designstruktur aufgrund des Spezifikationsgraphens aus Abbildung 6.24 dargestellt ist.

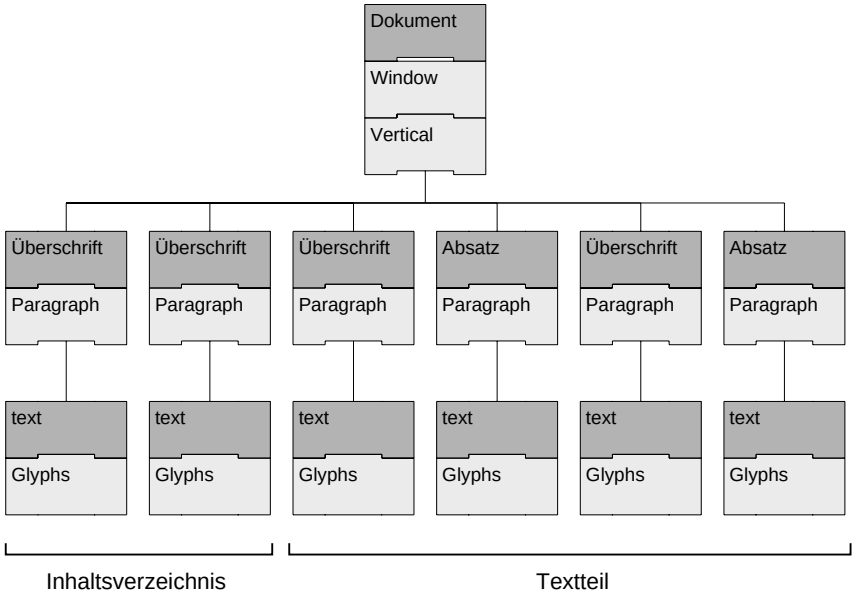

Abbildung 6.31: Designstruktur zu Abbildung 6.24 mit logischen Objekten: Die mit dargestellten logischen Objekte zeigen deutlich, dass die ersten beiden Paragraphen in der Designstruktur aufgrund der logischen Objekte `Überschrift` erstellt wurden und somit zur Realisierung des Inhaltsverzeichnisses dienen. Der dritte Paragraph wird zwar ebenfalls aufgrund eines logischen `Überschrift` Objektes verwendet, er realisiert aber bereits die erste Überschrift im Textteil.

An dieser Stelle ist es wichtig zu betonen, dass Objekte, die in [de:] an mehreren Stellen gleichzeitig angezeigt werden, in allen diesen Darstellungen immer im selben Zustand angezeigt werden. Ändert sich der Zustand eines Objektes aufgrund einer Interaktion des Nutzers mit einer Darstellung, so wird der Zustand gemäß dem *Model-View-Controller* Konzept von Smalltalk [17] im Model geändert und dann über den Controller an die Views weitergegeben. Die Grundlage dieses Konzeptes wurde bereits bei der Realisierung der Zustände der Spezifikationsobjekte auf den Seiten 187 und 199 verwendet.

Die Vorteile der Auswirkung dieses Model-View-Controller Konzeptes zeigen sich insbesondere bei der gleichzeitigen Verwendung des Hauptfensters mit dem Spezifikationsgraphen und dem Designstruktur-Fenster. Wird von einem Nutzer eines der sichtbaren Objekte z.B. im Spezifikationsgraphen manipuliert und deswegen sein Zustand geändert, so wird dies nicht nur in der Darstellung des Spezifikationsgraphens angezeigt, sondern auch unverzüglich in der Designstruktur. Dies gibt dem Nutzer ein direktes Feedback, welche Komponenten einer Spezifikationsstruktur wo in ihrer Designstruktur vorkommen und umgekehrt.

Die Objekteigenschaften

Mit Hilfe des Menüpunktes `Window - Object Properties` kann ein spezielles Objekteigenschaften-Fenster geöffnet werden. Mit seiner Hilfe können die Eigenschaften aller Objekte, die im Hauptfenster zur Verfügung stehen (logische Objekte und Spezifikationsobjekte), von den Nutzern begutachtet und falls nötig auch geändert werden. In einer Liste werden dazu je nach Objektart die Attribute und/oder Variablen, die das derzeit selektierte Objekt besitzt, angezeigt und falls möglich zur Bearbeitung angeboten.

Ist aktuell ein logisches Objekt selektiert, so werden die logischen Attribute, die es besitzt, in der Liste angezeigt. Sie stammen entweder aus dem eingelesenen logisch ausgezeichneten Beispieldokument, mit dessen Hilfe die Designspezifikation erarbeitet werden soll, oder sind mit Hilfe eines `SetLogAtt` Spezifikationsobjektes im Rahmen einer Anreicherung dem logischen Objekt zugefügt worden. Aus diesem Grund sind die logischen Attribute eines logischen Objektes mit Hilfe des Editors nicht editierbar. Bei Auswahl eines logischen Attributes aus dieser Liste wird jedoch der aktuell gültige Wert für dieses Attribut angezeigt.

Für den Fall eines aktuell selektierten Spezifikationsobjektes werden alle Spezifikationsatttribute angezeigt, die es besitzt. Bei Auswahl eines dieser Spezifikationsatttribute wird seine derzeit gültige Berechnungsvorschrift (der *statische* Wert eines Spezifikationsattributes) in einem Eingabefeld angezeigt. Diese Berechnungsvorschrift kann von den Nutzern bearbeitet werden. Parallel zu dieser statischen Berechnungsvorschrift für den Wert eines Spezifikationsattributes gibt es natürlich auch noch den dafür aktuell berechneten *dynamischen* Wert, der den Nutzern ebenfalls angezeigt wird. Er ergibt sich durch Auswertung des statischen Wertes mit Hilfe des Spezifikationsvariablenflusses[9]. Aufgrund der automatischen Berechnung kann der dynamische Wert natürlich nicht direkt geändert werden. Im Falle einer Mehrfachverwendung (siehe Seite 202) eines Spezifikationsobjektes in der Designstruktur kann der entsprechende dynamische Wert der n-ten Verwendung über ein Eingabefeld für n angefordert werden. Standardmäßig ist n in diesem Feld mit 1 vorbelegt.

Für den Fall, dass ein selektiertes Spezifikationsobjekt ein Auswertungsobjekt ist, kann dieses in der Auswahlliste Spezifikationsvariablen enthalten, die es manipulieren kann. Wie die Spezifikationsattribute können die zugeordneten Spezifikationsvariablen angewählt werden, damit die Nutzer ihren statischen Wert zur Bearbeitung in einem Eingabefeld zur Verfügung gestellt bekommen. Selbstverständlich wird auch hier wiederum der berechnete dynamische Wert in einem nichteditierbaren Feld angezeigt. Zu einem Auswertungsobjekt hinzugefügte Spezifikationsvariablen können mit Hilfe eines Buttons wieder entfernt werden. Hinzugefügt werden Spezifikationvariablen mit Hilfe des Fensters `Windows - Variables`, das eine Liste aller verfügbaren Spezifikationvariablen anzeigt. Zugleich kann in diesem Variablen-Fenster auch der initiale Wert (der Wert, der vor der Auswertung des Wurzelobjektes gesetzt wird) einer Spezifikationsvariablen für [de:] angegeben werden.

[9] Für nähere Information hierzu siehe auch 4.4.

6.2.8 Vorgehensmodell für die Arbeit mit [de:]

Für den qualifizierten und reibungslosen Ablauf der Arbeit mit [de:] ist es wie bei allen komplexen Systemen nötig, ein geeignetes Vorgehensmodell zur Verfügung zu haben. Dieses kann zwar wie üblich nicht als universelles Patentrezept zur Lösung aller mit einem System überhaupt lösbaren Aufgabenstellungen dienen. Es unterstützt jedoch die Nutzer bei der Realisierung von Lösungen für die alltäglichen Aufgabenstellungen, die einen großen Prozentsatz aller möglichen Fälle darstellen. Auch für einen ersten Einstieg in die Arbeit mit einem neuen System ist ein derartiges Vorgehensmodell als Leitfaden auf jeden Fall vorteilhaft.

Aufgrund der Erfahrungen, die sich während der Entwicklung des Designeditors [de:] und der Arbeit mit ihm ergeben haben, wird deshalb im Folgenden ein vierphasiges Vorgehensmodell für die Erstellung von Designspezifikationen mit Hilfe des Designeditors [de:] vorgeschlagen:

1. **Prinzipielles Layout für einzelne Views festlegen:**
   Die Erstellung des prinzipiellen Layouts für die einzelnen *grundlegenden* Views, die in einer Designspezifikation verwendet werden, erfolgt in [de:] durch das Einbringen von Spezifikationsobjekten in Formatierungen durch Drag and Drop Operationen. Die verwendeten Spezifikationsobjekte legen, wie bereits beschrieben, mit Hilfe der durch sie verkörperten Funktionalität schon im Wesentlichen die Verarbeitung der betroffenen logischen Objekte fest.

2. **Details für einen View festlegen:**
   Nachdem in der ersten Phase die nötigen Spezifikationsobjekte in die Formatierungen eingebracht worden sind, kann nun in der zweiten Phase damit begonnen werden, die Details der anzuwendenden Verarbeitungen für die logischen Objekte festzulegen. Dies geschieht zum einen, indem mit Hilfe der Auswertungsobjekte Berechnungsvorschriften für die in der Designspezifikation vorkommenden und verwendeten Spezifikationsvariablen aufgestellt werden. Zum anderen sind in dieser Phase auch die Berechnungsvorschriften für die Spezifikationsattribute der Spezifikationsobjekte festzulegen, um ihr allgemeines Verhalten an die jeweilige spezifische Aufgabenstellung anzupassen.

3. **Komposition von Views:**
   Nachdem nun die einzelnen grundlegenden Views, die für das Design eines Dokumentes verwendet werden sollen, festgelegt wurden, können diese in der Kompositionsphase zu neuen Views zusammengesetzt werden. Diese Phase kann selbstverständlich den Rückschritt in Phase 2 nötig machen, um auch für die zusammengesetzen Views die Detailsteuerung zu ermöglichen. Es ist offensichtlich, dass in der dritten Phase erstellte Views wiederum in anderen Views verwendet werden können.

4. **Test:**
   Selbstverständlich ist es nach Abschluß der ersten drei Phasen, nach denen theoretisch die Erstellung einer Designspezifikation abgeschlossen ist, nötig, diese auch noch ihre *Aufgabenerfüllung* hin zu überprüfen. Dies ist jedoch ein heikler Punkt, dessen Aufwand wie bei der Software-Entwicklung nur schwer abzuschätzen ist.

Es ist klar, dass die einzelnen, oben aufgeführten vier Phasen nicht streng sequentiell nacheinander im Rahmen der Erstellung einer Designspezifikation abgearbeitet werden. Dies wurde bereits in Phase drei, der Komposition von Views, angedeutet. Je nach Komplexität der Aufgabe werden potentielle Nutzer des Systems die vier Phasen in mehr oder weniger kurzen Zyklen durchlaufen, um somit inkrementell an ihr Ziel zu gelangen. Das hier vorgestellte Vorgehensmodell ist somit weniger als ein exakter Rahmen zu betrachten, sondern vielmehr als eine Anleitung für einen ersten systematischen Einstieg in die Arbeit mit [de:] zu verstehen.

6.2.9 Das Testen und *Debuggen* von Designspezifikationen

Wie in allen produktiven Prozessen ist es auch nach der (teilweisen) Erstellung einer Designspezifikationen nötig, sie einer Kontrolle zu unterziehen. Hierbei gilt es zu überprüfen, ob sie auch tatsächlich das spezifiziert, was ihr Ersteller *im Sinn* hat. Dabei geht es zunächst in erster Linie noch gar nicht unbedingt um die Suche von kleinen Detailfehlern, die unter Umständen nur in gewissen *esoterischen* Situationen entstehen. Vielmehr gilt es in einer ersten Phase zu testen, ob das prinzipielle Verhalten einer Designspezifikation korrekt ist.

Der Test einer Designspezifikation

Bereits diese erste grundlegende Eigenschaft einer Designspezifikation, im Wesentlichen das zu tun, was sie soll, ist unter Umständen nur schwer nachzuvollziehen. Die Designregeln, die zusammen eine Designspezifikation ergeben, werden aufgrund eines ereignis- und strukturgesteuerten Verfahrens zum Aufbau einer Designstruktur ausgewählt und verwendet. Eine Designspezifikation ist somit nicht direkt mit einem normalen Software-Programm zu vergleichen, das fest vorgegeben ist und für einen Test einfach gestartet wird: Das Programm ist ein schon fertiges Produkt. Eine Designspezifikation hingegen ist im Vergleich hierzu nur ein *Halbfertigprodukt*. Sie dient dazu, eine Art von Programm, die Designstruktur eines Dokumentes, zu berechnen. Eine Designspezifikation befindet sich somit auf einer abstrakteren Ebene als normale Software-Programme.

Eine Möglichkeit, wie eine Designspezifikation bereits während ihrer Erstellung getestet werden kann, stellt der Spezifikationsgraph und die Verwendung der Anzeige der Designstruktur dar: Das Prozessmodell für [de:], das Pipelinesystem, ermöglicht den Nutzern schnell einen tiefgreifenden Einblick in die Auswirkungen einer Designspezifikation. Wie bei den Testläufen von Software-Programmen ist es aber auch hier wünschenswert, einen echten Testlauf einer Designspezifikation durchführen zu können: Die Anzeige des Resultats einer Designapplikation auf ein gegebenes logisch ausgezeichnetes Dokument in Form einer Bildschirmpräsentation. Dies ist im [es:] System aufgrund der Realisierung eines geeigneten Dokumentformatierers [ef:] möglich. Die Details hierzu werden in Kapitel 7 besprochen.

Das Debuggen einer Designspezifikation

Wenn nun tatsächlich ein Problem in einer Designspezifikation auftritt, so gibt es zwei mögliche Punkte, an denen der zugrundeliegende Fehler von einem Designer begangen worden sein kann.

Dies ist zum einen die Verwendung einer *falschen* Kombination von Spezifikationsobjekten in einer Designregel und zum anderen die *fehlerhafte* Setzung von Spezifikationsvariablen und Spezifikationsattributen für Spezifikationsobjekte.

Den Nutzern von [de:] müssen deshalb geeignete Werkzeuge zur Verfügung gestellt werden, mit deren Hilfe diese oben aufgeführten zwei Fehlerquellen in einer Designspezifikation analysiert werden können. Diese können dann sogar nicht nur zur Fehlersuche in einer Designspezifikation eingesetzt werden, sondern auch zur Übernahme und zum Verstehen von Designspezifikationen, die von anderen Designern erstellt worden sind.

Während das geeignete Werkzeug für die Untersuchung der ersten Fehlerquelle (*falsche* Kombination von Spezifikationsobjekten) der Spezifikationsgraph ist, sind für die Analyse von Fehlern im Bereich der Wertsetzung für Spezifikationsvariablen und Spezifikationsattribute bisher noch keine geeigneten Werkzeuge vorgestellt worden. Die zwei Grundlagen hierfür werden nun im Folgenden aufgezeigt:

- **History-Analyse:**
  Das Ziel der History-Analyse ist die Klärung der Frage: *Von wem hängt der betrachtete Wert eines Spezifikationsattributes oder einer Spezifikationsvariablen ab?*

- **Dependency-Analyse:**
  Bei der Dependency-Analyse ist die Betrachtungsweise genau umgekehrt zu obiger Situation. Die Fragestellung lautet hier: *Wer ist von einem betrachteten Attribut oder einer betrachteten Spezifikationsvariable abhängig?*

Die Werte von Spezifikationsvariablen, die zur Setzung der Werte von Spezifikationsattributen in einer Formatierung verwendet werden können, sind durch das Vorkommen von Auswertungsobjekten in einem Spezifikationsgraphen beliebig manipulierbar. Die Stellen, an denen bestimmte Änderungen der Spezifikationsvariablen stattfinden, sowie die Art der Änderungen, sind aber in der bisherigen Darstellung eines Spezifikationsgraphens nicht sichtbar. Man sieht zwar `Inheritance` und `Modifier` Auswertungsobjekte als solche, nicht aber, in welcher Art und Weise sie auf die Spezifikationsvariablen wirken[10]: Eine derartige Anzeige würde aufgrund der mannigfaltigen Beziehungen innerhalb einer Designspezifikation den Spezifikationsgraph vollkommen *überfluten* und unübersichtlich machen. Zusätzlich können auch noch die Attribute der logischen Objekte zur Wertsetzung beitragen. Zusammengefasst gilt also: Man kann in einem Spezifikationsgraph sehen, dass *irgendwelche* Spezifikationsvariablen auf *irgendeine* Weise verändert werden und *irgendwo* verwendet werden, nicht jedoch, *welche* und auf *welche* Weise.

In [de:] lässt sich auch immer nur ein einziges `Object Properties`-Fenster zur gleichen Zeit öffnen – eine Eigenschaft, die Aufgrund der Größe des Fensters und dessen primärer Aufgabe, der Wertspezifizierung für Eigenschaften von Objekten, auch sinnvoll ist. Somit ist es in der Praxis beim Umgang mit [de:] bisher nicht möglich, zur Verfolgung von Spezifikationsvariablen- und Spezifikationsattributabhängigkeiten mit Hilfe mehrer Fenster alle notwendigen Informationen gleichzeitig im Blickfeld zu haben.

[10] Dies ist bisher allein im `Object Properties`-Fenster für *ein* betroffenes Spezifikationsobjekt sichtbar.

Tatsächlich taucht aber in der Praxis häufig die Fragestellung „Warum hat die Spezifikationsvariable bzw. das Spezifikationsattribut `x` an dieser Stelle im Spezifikationsgraphen den Wert `y`?“ auf. Und dies naheliegender Weise insbesondere dann, wenn der Umfang der Designspezifikation zunimmt. Nicht minder interessant ist aber auch die umgekehrte Fragestellung, „An welchen Stellen ändern sich Werte in der Designstruktur, wenn ich einen Ausdruck zur Berechnung hier an einer Stelle ändere?“. Dies ist die Frage nach den oft unerwarteten Seiteneffekten, die Änderungen nach sich ziehen können.

Somit ergibt sich für [de:] die Aufgabe, den Nutzern mit Hilfe zweier geeigneter graphischer Darstellungen Anworten auf diese im Rahmen der Erstellung einer Designspezifikation immer wieder auftauchenden zwei Fragestellungen liefern zu können.

History-Analyse

Die History-Analyse in [de:] dient dazu, den Bearbeitern eines Spezifikationsgraphen in geeigneter Form die *vollständige* Vorgeschichte des Wertes einer Spezifikationsvariablen- oder eines Spezifikationsattributes (im Folgenden auch Eigenschaft genannt) an einer von ihnen vorgegebenen Stelle im Spezifikationsgraphen graphisch darzustellen. Vollständig bedeutet dabei, dass bei der Berechnung der History-Analyse alle Spezifikationsvariablen und logischen Attribute betrachtet werden, die zu der Wertberechnung der zu untersuchenden Eigenschaft des Objektes beitragen – eine Forderung die rekursiv zu verstehen ist. Folgendes Beispiel soll die Forderung der vollständigen Vorgeschichte verdeutlichen: Ist ein Spezifikationsattribut `x` für ein ausgewähltes Layoutobjekt an der betrachteten Stelle als `x:=a+b` festgelegt und ergibt sich weiter oben im Spezifikationsgraphen `a:=c+d`, so ist `x` insgesamt von `a`, `b`, `c` und `d` abhängig. Abbildung 6.32 zeigt dies in einer Baumdarstellung auf.

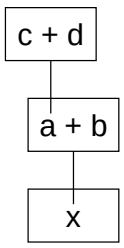

Abbildung 6.32: Vorgeschichtsbaum für die Berechnung des Wertes eines Spezifikationsattributes: `x` ist von `a`, `b`, `c` und `d` abhängig.

Neben einer Visualisierung dieser vollständigen Vorgeschichte aufgrund der statischen Berechnungsvorschriften soll es zur Förderung der Verständlichkeit in der Anzeige der History-Analyse aber gleichzeitig auch noch möglich sein, die sich dynamisch ergebenden Zwischenergebnisse der beteiligten statischen Berechnungsvorschriften (die Boxen im Vorgeschichtsbaum) betrachten zu können.

Aus diesem Grund wird in [de:] zur Anzeige im History-Analyse-Fenster eine Darstellung gewählt, wie sie in Abbildung 6.33 zu sehen ist. In dieser zeilenorientierten Listendarstellung sind untereinander die Objekte (logische Objekte und Spezifikationsobjekte) aufgeführt,

die an der Beeinflussung des selektierten zu untersuchenden Wertes für ein Spezifikationsobjekt beteiligt sind. Ganz links sind jeweils die Namen dieser beteiligten Spezifikationsobjekte sowie rechts daneben die sich dynamisch ergebenden Zwischenergebnisse und die Namen der Eigenschaften aufgeführt. Ganz rechts in der Darstellung ist innerhalb dieser Liste eine Liniengraphik zu finden, die den in Abbildung 6.32 exemplarisch eingeführten Vorgeschichtsbaum eindeutig und übersichtlich in Zeilenform darstellt. Die chronologische Reihenfolge der Berechnung lässt sich in dieser Darstellung durch die Zeilenfolge (bzw. Objektfolge) nachvollziehen. Ebenso sind Mehrfachabhängigkeiten gut zu sehen und zu quantifizieren (anhand der Anzahl der Linienknotenpunkte in einer Zeile). Die Parallelauswertung innerhalb eines Objektes (siehe 4.4.5 auf Seite 122) wird durch Zusammenfassung mehrerer Zeilen in der Namensspalte kenntlich gemacht (z.B. bei dem `Inheritance` Auswertungsobjekt in Abbildung 6.33).

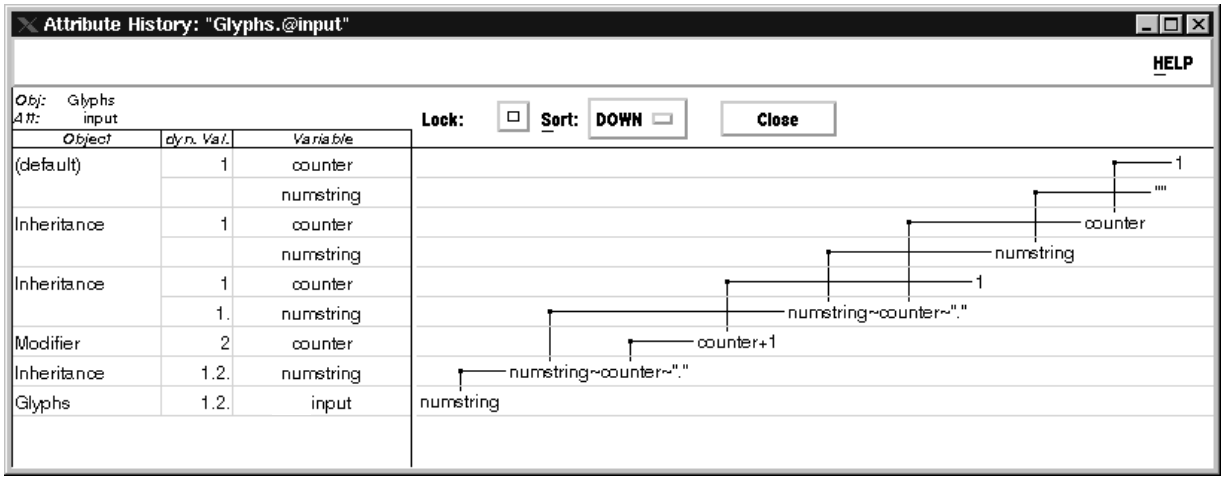

Abbildung 6.33: Das History-Analyse Fenster: Die hier dargestellte History-Analyse ist für das `input` Attribut des mit [a] bezeichneten Spezifikationsobjektes aus Abbildung 6.28 durchgeführt worden.

Zum Aufruf des History-Analyse-Fensters wählt man zuerst die zu verfolgende Spezifikationsvariable bzw. das zu verfolgende Spezifikationsattribut für ein Spezifikationsobjekt im `Object Properties`-Fenster an, bevor man über den Menüpunkt `Windows - Attribute History` das Fenster öffnet. Es können beliebig viele Instanzen dieses Fensters gleichzeitig geöffnet sein. Ist zum Zeitpunkt des Öffnens eines `Attribute History`-Fensters kein Spezifikationsobjekt selektiert, so erscheint ein leeres Fenster. Mit Hilfe mehrerer Oberflächenkomponenten kann das Verhalten eines `History Analyse`-Fensters gesteuert werden. Diese sind im Folgenden aufgeführt:

Lock:
Wählt man `Lock`, so wird in diesem Fenster stets die Attribute-History für die zum Zeitpunkt des Fixierens darin angezeigte Spezifikationsobjekt/Eigenschaft-Kombination angezeigt, auch wenn zwischenzeitlich ein anderes Spezifikationsobjekt bzw. eine andere Eigenschaft angewählt wird. Ist `Lock` nicht aktiv, so wird in diesem Fenster immer die gerade aktuell selektierte Objekt/Eigenschaft-Kombination angezeigt und somit immer der Fensterinhalt bei Neuselektion eines Objektes oder einer Eigenschaft ausgetauscht.

Sortierung:
Mit diesem Popup-Menü legt man die Sortierrichtung fest, in der die Listendarstellung angezeigt wird: Mögliche Werte sind vom gewählten Spezifikationsobjekt aus in Richtung Wurzel des Spezifikationsgraphens oder umgekehrt.

Scrolling:
Durch Drücken der mittleren Maustaste im Anzeigebereich kann der Fensterinhalt gescrollt werden. Der linke Anzeigebereich (Objekt, dynamischer Wert und Eigenschaftsname) kann dabei nur in vertikaler Richtung gescrollt werden, der rechte Anzeigebereich sowohl in vertikaler als auch horizontaler Richtung. Das vertikale Scrolling findet notwendigerweise immer synchronisiert zwischen beiden Bereichen statt.

Interaktionsmöglichkeiten:
Beim Überstreichen des Objektnamens in den einzelnen Zeilen wird gemäß dem Model-View-Controller Konzept eine entsprechende Aktualisierung in allen von dieser Aktion betroffenen Anzeigen im [de:] System vorgenommen. Der umgekehrte Weg ist ebenfalls möglich: wird ein Objekt in einer anderen Darstellung überstrichen, so wird dies auch in dieser Anzeige dargestellt. Ein Klick mit der linken Maustaste auf einen Objektnamen selektiert das entsprechende Objekt im System.

Dependency-Analyse

Die Dependency-Analyse ist funktional genau das Gegenteil der History-Analyse. Hier soll ein Designer bei der analytischen Aufgabe unterstützt werden, zu erkennen, welche logischen und welche Spezifikationsobjekte mit welchen Eigenschaften in einer Designspezifikation von einer betrachteten Eigenschaft eines ausgewählten logischen Objektes oder Spezifikationsobjektes abhängig sind. Dieses Wissen wird von einem Designer beispielsweise dann benötigt, wenn er sich nicht sicher ist, an welchen Stellen in einer Designspezifikation bestimmte von ihm schon eingeführte Eigenschaften verwendet werden.

In [de:] gibt es genau sieben Möglichkeiten, wann ein logisches Objekt oder ein Spezifikationsobjekt von einer Eigenschaft eines logischen Objektes oder eines Spezifikationsobjektes abhängig ist. Diese sind im Folgenden aufgelistet:

1. **Alias-Spezifikationsobjekt:**
   Ein Spezifikationsobjekt wird aufgrund derselben Designregel wie das Spezifikationsobjekt angewendet, für das die Dependency-Analyse durchgeführt wird. Somit wirken sich natürlich *alle* Änderungen, die am untersuchten Spezifikationsobjekt ausgeführt werden, auch an diesem Spezifikationsobjekt aus: Das betrachtete Spezifikationsobjekt ist lediglich wie das untersuchte Spezifikationsobjekt ein Alias für das bezogene Spezifikationsobjekt in der Designregel, das bei einer Bearbeitung geändert wird.

2. **Früheres** `Inheritance` **Auswertungsobjekt:**
   Ein Spezifikationsobjekt ist von einem früheren `Inheritance` Auswertungsobjekt abhängig, wenn dieses in der Designstruktur als ein Vorfahre angeordnet ist und eine Spezifikationsvariable modifiziert, die das abhängige Spezifikationsobjekt verwendet.

3. **Früheres** `Modifier` **Auswertungsobjekt:**
   Ein Spezifikationsobjekt ist von einem früheren `Modifier` Auswertungsobjekt abhängig, wenn dieses in der Auswertung vor ihm kommt und eine Spezifikationsvariable modifiziert, die das abhängige Spezifikationsobjekt verwendet.

4. **Früheres** `Blocker` **Auswertungsobjekt:**
   Alle Spezifikationsobjekte, die unter einem `Blocker` Auswertungsobjekt vorkommen (siehe Seite 125) sind von diesem Spezifikationsobjekt abhängig: Es wirkt wie eine `Inheritance` Auswertungsobjekt auf alle verfügbaren Spezifikationsvariablen, ohne sie zu ändern.

5. **Früheres** `Repeater` **Auswertungsobjekt:**
   Alle Objekte, die unter einem `Repeater` Auswertungsobjekt (siehe Seite 126) vorkommen und auf eine Spezifikationsvariable dieses `Repeater` Auswertungsobjektes zugreifen, sind von diesem abhängig.

6. **Früheres logisches Objekt:**
   Ein Spezifikationsobjekt, das auf einen verfügbaren Wert eines logischen Objektes zugreift, ist von diesem logischen Objekt abhängig.

7. **Späteres** `SetLogAtt` **Auswertungsobjekt:**
   Das logische Objekt, für das in seiner Formatierung ein `SetLogAtt` Auswertungsobjekt vorkommt, ist von diesem `SetLogAtt` Auswertungsobjekt abhängig.

Für die graphische Darstellung der Dependency-Analyse aller von einer Objekt/Attribut-Kombination abhängigen Objekt/Attribut-Kombinationen wurde der Konsistenz wegen eine ähnliche Listenform wie für die History-Analyse verwendet. Abbildung 6.34 zeigt die Darstellung einer Dependency-Analyse. Sie ist auf der linken Seite identisch zur History-Analyse aufgebaut und enthält auf der rechten Seite jeweils die Begründung für die Abhängigkeit der einzelnen links aufgeführten Objekt/Attribut-Kombinationen von der Ausgangs Objekt/Attribut-Kombination.

Neben dieser detaillierten Anzeigemöglichkeit der Dependency-Analyse in einem speziellen Fenster gibt es in [de:] noch die Möglichkeit, die abhängigen Objekte einer Objekt/Attribut-Kombination im Spezifikationsgraphen anzeigen zu lassen. Dabei werden alle Objekte, die von *irgendeiner*[11] der Eigenschaften desjenigen Objektes, auf dem der Mauszeiger aktuell steht, abhängig sind, mit einem dicken schwarzen Balken am rechten Rand gekennzeichnet. Auf diese Weise können die Nutzer direkt im Spezifikationsgraph erkennen, welche Objekte bei einer Änderung des vorselektierten Objektes betroffen wären. Diese Anzeigeoption im Spezifikationsgraphen wird mit Hilfe des Menüpunktes `Show - Show Dependencies` aktiviert bzw. mit `Show - Hide Dependencies` deaktiviert.

Zum Aufruf des Dependency-Analyse-Fensters wählt man zuerst die gewünschte Objekt/Attribut-Kombination an, für die die abhängigen Objekt/Attribut-Kombinationen angezeigt werden sollen. Dann öffnet man über den Menüpunkt `Windows - Dependency List` das

[11] In einem Dependency-Analyse Fenster werden immer nur die bezüglich *einer einzelnen* Eigenschaft abhängigen Objekte angezeigt.

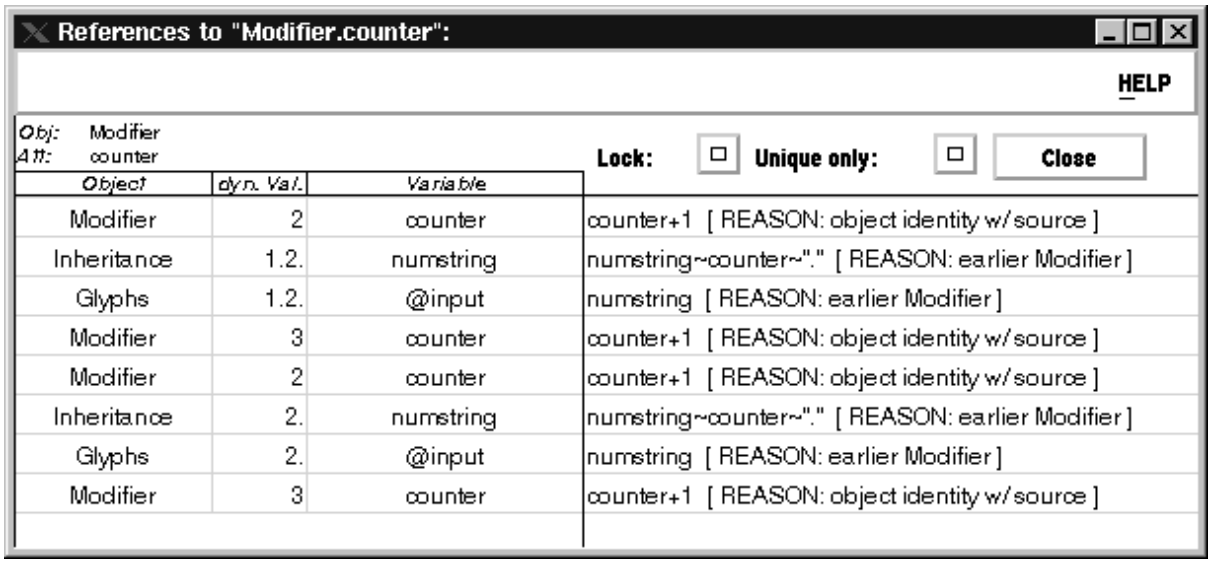

Abbildung 6.34: Das Dependency-Analyse Fenster: Die hier dargestellte Dependency-Analyse ist für die `counter` Spezifikationsvariable des mit b bezeichneten `Modifier` Auswertungsobjektes aus Abbildung 6.28 durchgeführt worden.

Fenster. Es können beliebig viele Instanzen dieses Fensters gleichzeitig geöffnet sein. Ist zum Zeitpunkt des Öffnens keine Objekt/Attribut-Kombination selektiert, so erscheint ein leeres Fenster. Mit Hilfe mehrerer Oberflächenkomponenten kann das Verhalten eines Dependency-Analyse-Fensters gesteuert werden. Diese sind im Folgenden aufgeführt:

Unique only:
Es ist möglich, sich *alle* betroffenen Objekte des Spezifikationsgraphen anzeigen zu lassen, also auch Aliases (Option nicht aktiviert), oder die Anzeige auf nur jeweils einen Repräsentanten zu beschränken (Option aktiviert).

Lock:
Wählt man `Lock`, so wird in diesem Fenster stets die Dependency-Analyse für die zum Zeitpunkt des Fixierens darin angezeigte Objekt/Attribut-Kombination angezeigt, auch wenn zwischenzeitlich etwas anderes ausgewählt wird. Ist `Lock` nicht aktiv, so wird in diesem Fenster immer die Dependency-Analyse für die gerade aktuell ausgewählte Objekt/Attribut-Kombination angezeigt und somit der Fensterinhalt ausgetauscht.

Scrolling:
Durch Drücken der mittleren Maustaste im Anzeigebereich kann der Fensterinhalt wie in [de:] üblich gescrollt werden. Näheres hierzu auf Seite 222.

Interaktionsmöglichkeiten:
Beim Überstreichen der Dependency-Analyse mit der Maus können diverse Aktionen im System ausgelöst werden. Näheres hierzu auf Seite 222.

6.2.10 Compoundobjekte

Compoundobjekte stellen eine Zusammenfassung einer beliebigen Formatierung zu einem einzigen, neuen Spezifikationsobjekt dar. Sie dienen der vereinfachten Spezifikation, indem sie häufig verwendete und benötigte Spezifikationsobjekt-Kombinationen als neue Bausteine zur Verfügung stellen. Diese können von den Nutzern des Systems in gleicher Weise wie die bisher eingeführten Spezifikationsobjekte verwendet werden. Um sie in verschiedenen Designspezifikationen gleichzeitig verwenden zu können, werden sie nicht in einer Designspezifikation gespeichert, sondern [de:]-global verwaltet und gespeichert.

Definieren eines neuen Compoundobjektes

Um ein neues Compoundobjekt zu definieren, selektiert man im aktuellen Spezifikationsgraphen ein *Start-Spezifikationsobjekt* in einer Formatierung für ein logisches Objekt. Dann wählt man den Menüpunkt `Functions - Make Compound`, wodurch ein neues Compoundobjekt im linken Bereich erstellt wird. Dieses enthält den kompletten Unterbaum inklusive des Start-Spezifikationsobjektes (linke und rechte Geschwister eingeschlossen). Für jeden in der definierenden Objektkombination enthaltenen Konnektor wird dabei ein Objekteingang für das Compoundobjekt erzeugt.

Zuweisen von Gültigkeitsklassen an die Attribute

Öffnet man das Object Properties-Fenster für ein Compoundobjekt im linken Bereich (nicht im Spezifikationsgraphen!), so kann mit Hilfe eines Optionsmenüs einer zuvor angewählten Spezifikationsvariablen der gewünschte Gültigkeitsbereich (siehe 4.7.1 auf Seite 141) zugewiesen werden. Die Bedeutungen sind:

LOCAL Der Wert der betreffenden Eigenschaft berechnet sich durch den hier angegebenen Ausdruck; Änderungen durch Auswertungsobjekte im Compoundobjekt haben keine Auswirkungen außerhalb des Compoundobjektes.

GLOBAL Der Wert der betreffenden Eigenschaft berechnet sich aus der gleichnamigen Spezifikationsvariable; Änderungen durch Auswertungsobjekte auf die Spezifikationsvariable im Compoundobjekt haben globale Auswirkungen.

Sichern und Laden der Compoundobjekt-Definitionen

Die Definitionen von Compoundobjekten werden automatisch bei Beendigung des Systems, beim Sichern der Optionen und beim Sichern der Designspezifikation im aktuellen Verzeichnis in der Datei `compounds.tool` gesichert. Beim Start des Systems werden sie automatisch aus dieser Datei wieder eingelesen.

Anwendung eines Compoundobjektes

Compoundobjekte können wie alle anderen Spezifikationsobjekte auch per Drag and Drop Operationen im Rahmen einer Designspezifikation verwendet werden. Im Object Properties-Fenster für ein Compoundobjekt kann dabei für alle Attribute der Klasse Local ein neuer Ausdruck spezifiziert werden. Für Attribute der Klasse Global ist dies nicht möglich. Ebenso kann auch für ein Compoundobjekt im Spezifikationsgraphen die Gültigkeitsbereichklasse selbst nicht geändert werden. Weiterhin wird für alle Spezifikationsvariablen mit dem Gültigkeitsbereich Global, die ein Compoundobjekt beinhaltet, angezeigt, ob das Compoundobjekt wie ein `Inheritance` oder ein `Modifier` Auswertungsobjekt auf den Spezifikationsvariablenfluss bezüglich dieser Spezifikationsvariablen wirkt. Dies ist nötig, da ein Compoundobjekt als einziges Spezifikationsobjekt nicht bereits von außen verrät, wie es auf die einzelnen in ihm vorkommenden Spezifikationsvariablen wirkt.

6.2.11 Umgebungsprädikate

In [de:] wird immer dann, wenn ein erstes Spezifikationsobjekt an ein logisches Objekt angefügt wird, eine neue Default-Designregel mit einem allgemein gültigen Umgebungsprädikat für das betroffene logische Objekt angelegt. Unter Umständen kann es aber auch nötig sein, für ein spezielles logisches Objekt eine kontextabhängige Verarbeitung zu spezifizieren. Das Vorgehen hierzu wird im Folgenden beschrieben.

Falls ein logisches Objekt o im Spezifikationsgraph markiert ist, ist es durch Anwahl des Menüpunktes `Functions - Create New Context` möglich, für dieses logische Objekt o eine neue Designregel mit einem Umgebungsprädikat aufzustellen; das logische Objekt o wird dabei zu dem Basisknoten des aufzustellenden T-Kontextes. Für den Fall, dass für das logische Objekt o zu diesem Zeitpunkt im Spezifikationsgraph bereits eine Designregel angezeigt wurde, wird die angezeigte Formatierung in die neue Designregel übernommen.

Die Eingabe des strukturellen T-Kontextes (siehe 5.3.2) für das gegebene logische Basisobjekt o erfolgt einfach durch Anklicken der gewünschten logischen Objekte, die für die in Abschnitt 5.4.3 beschriebene Berechnung des T-Kontextes verwendet werden sollen. Die gewählten Logischen Objekte werden in einer speziellen Farbe dargestellt, die kennzeichnet, dass die logischen Objekt zur Kontextberechnung verwendet werden sollen (siehe Abbildung 6.35). Werden logische Objekte, die ausserhalb der möglichen T-Umgebung des logischen Objektes o liegen, selektiert, so werden sie bei der Berechnung des T-Kontextes ignoriert.

Für jeden Kontext muss im Einstellungfenster ein Name vorgegeben werden, der nach Möglichkeit sinnvoll seine Aufgabe bezeichnen sollte. Neben diesem T-Kontext kann weiterhin auch ein boolescher Ausdruck angegeben werden, der jedoch ausschließlich aus logischen Attributen aufgebaut sein darf.

Nach Verlassen des Kontextmodus (durch Anklicken des dafür zur Verfügung gestellten Buttons) wird automatisch eine neue Designregel erstellt, die das spezifizierte Umgebungsprädikat enthält. Anschließend wird, basierend auf der neuen Menge von Designregeln, eine neue Designapplikation durchgeführt.

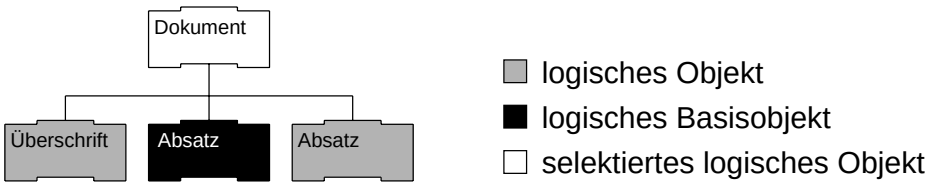

Abbildung 6.35: Spezifikationsgraph während einer Kontextspezifikation: Das abgebildete Beispiel zeigt die T-Kontextspezifikation des Beispiels aus Abbildung 5.11 auf Seite 174.

Soll ein bereits bestehendes Umgebungsprädikat geändert werden, so ist ein logisches Objekt anzuklicken, für das das gewünschte Umgebungsprädikat gültig ist. Es kann nun der Menüpunkt `Functions - Edit Current Context` gewählt werden, um sowohl den T-Kontext als auch den booleschen Ausdruck zu ändern. Bestehende T-Kontexte können über den Menüpunkt `Functions - Delete Current Context` auch wieder gelöscht werden.

Kapitel 7

Der Dokumentformatierer und Betrachter [ef:]

Im Kapitel 6 dieser Arbeit wurden die Werkzeuge [pe:] und [de:] vorgestellt, mit deren Hilfe Designer in einem ganzheitlichen Prozess Designspezifikationen gemäß der [em:] Designspezifikationsmethodik erstellen können. Die in diesem Kapitel vorgestellte Komponente [ef:] rundet nun die Menge der im Rahmen des [es:] Systems zur Verfügung gestellten Werkzeuge ab. Mit seiner Hilfe wird es den Designern ermöglicht, das Ergebnis einer Designapplikation anschaulich zu betrachten und zu überprüfen: [ef:] ist ein integrierter Formatierer und Betrachter von logisch ausgezeichneten Dokumenten, die mit Hilfe von [pe:] in eine Designstruktur überführt worden sind.

7.1 Motivation

An dieser Stelle der Arbeit gilt es zu begründen, warum ein neuer integrierter Dokumentformatierer und Betrachter, der im Folgenden mit [ef:] bezeichnet wird, entwickelt wird. Die drei wesentlichen Argumente hierzu sind in folgender Aufzählung ausgeführt:

- **Gründe basierend auf der Designspezifikationsmethodik [em:]:**
  Die in dieser Arbeit vorgestellte Designspezifikationsmethodik [em:] und das Ergebnis der Anwendung einer in ihr spezifizierten Designspezifikation auf ein logisch ausgezeichnetes Dokument, die Designstruktur, besitzen eine funktionale Mächtigkeit, die bisher verfügbare Dokumentformatierer nicht verarbeiten und umsetzen können.
- **Gründe, die auf einer allgemeinen Untersuchung von Dokumentformatierern und Betrachtern beruhen:**
  Dokumentformatierer sind komplexe Software-Produkte, die:
  - gemäß ihrer Aufgabenstellung jeweils nur gewisse Aufgabengebiete mit perfekten Ergebnissen abdecken. Umfassende Systeme, die alle im Bereich der Dokumentformatierung auftretenden Aufgabenstellungen lösen können, gibt es bisher nicht.

  - monolithisch (sämtliche Funktionalität der Dokumentformatierer wird in einem zentralen Programmblock realisiert) und daher nur schwer an neue Aufgabenstellungen anzupassen sind. Aufgrund der langen Entwicklungszeiten und ständigen Hinzufügung von Funktionalität wird es in diesen Systemen immer schwieriger, sie ohne das Auftreten von unerwünschten Nebeneffekten und globalen Codeänderungen warten und ergänzen zu können (siehe hierzu z.B. [35]).
  - insbesondere auch aufgrund des vorherigen Punktes beim Aufkommen neuer Paradigmen im Bereich der Dokumentverarbeitung vollkommen neu realisiert werden müssen. Ein besonders deutliches Beispiel hierfür ist das WWW mit eigens für es entwickelten Browsern, die aufgrund ungenügender Funktionalität und unzureichenden Modellen und Konzepten mehrfach neu implementiert werden mussten (siehe hierzu z.B. [41]). Zwar ergibt sich bei dieser ständigen Weiterentwicklung oft eine funktionelle Erweiterung der entstehenden Systeme, sie greifen aber meist nicht auf die qualitativ hochwertigen Lösungen von bereits bestehenden Systemen zurück.

- **Gründe aus Forschung und Lehre:**
  Zum einen sind bestehende Dokumentformatierer und Betrachter derart umfassende und komplexe Software, dass sie kaum sinnvoll in Forschung und Lehre eingesetzt bzw. weiterentwickelt werden können. Zum anderen muss, bevor in einem neuen Dokumentformatierer und Betrachter neue Algorithmen zur Lösung von speziellen Aufgaben realisiert werden können, zunächst zeitaufwendig eine komplexe und umfassende Software-Basis implementiert werden, in der viele technische Details (Fontverwaltung, Druckersteuerung, Benutzungsoberfläche) berücksichtigt werden müssen.

Mit obigen Punkten ergibt sich eine schlüssige Begründung für die Entwicklung der [ef:] Komponente. Die Zielsetzung bei ihrer Entwicklung sind, kurz zusammengefasst, die Erstellung eines Konzeptes und dessen prototypische Implementierung für einen Dokumentformatierer und Betrachter, der Designstrukturen der [em:] Designspezifikationsmethodik verarbeiten kann, obige aufgeführte Missstände von bestehenden Dokumentformatierern und Betrachtern überwindet und schließlich auch in der Forschung und Lehre weiterentwickelt und verwendet werden kann.

7.2 Anforderungsanalyse

Wie in der Motivation zu diesem Kapitel festgehalten wurde, ist seine Zielsetzung die Entwicklung einer fundierten Grundlage für einen integrierten Dokumentformatierer und Betrachter, der den in der Motivation genannten Forderungen entspricht. Es muss somit eine Analyse durchgeführt werden, um diejenigen Punkte herauszufinden, die es bei einem derartigen Unterfangen zu berücksichtigen gilt. Die Ergebnisse dieser Untersuchung der in der Motivation genannten Schwachpunkte sind in folgender detaillierter Liste zu finden:

- Es gilt die einzelnen Funktionalitäten bestehender Dokumentformatierer und Betrachter in Modulen zu kapseln und mit einem definierten Interface zu Wiederverwendung zu versehen.
- Die einzelnen derart definierten Module dürfen keine globalen Seiteneffekte haben.
- Die Module müssen beliebig wiederverwendbar und kombinierbar sein.
- Das entstehende System insgesamt muss erweiterbar sein. Der Aufwand dabei muss direkt proportional zum erzielten Effekt sein und darf nur von diesem abhängen. Bereits bestehende Module dürfen von einer Erweiterung nicht direkt betroffen sein.
- Es sollen verschiedene Medien (z.B. Papier und Bildschirm) als Präsentationsschnittstelle unterstützt werden.
- Es sollen beliebige Interaktionsmöglichkeiten (z.B. Scrolling und Linking) unterstützt werden.
- Das System soll speichereffizient und schnell sein.
- Das System soll hohen typographischen Anforderungen gerecht werden können.

7.3 Architektur

Basierend auf den soeben präsentierten Ergebnissen der Anforderungsanalyse gilt es nun, eine geeigente Systemarchitektur für den zu entwickelnden Dokumentformatierer und Betrachter [ef:] zu entwickeln. Betrachtet man hierzu die einzelnen Punkte der Anforderungsanalyse, so sieht man, dass eine softwaretechnische Umsetzung der in der Designspezifikationsmethodik [em:] eingeführten Layoutobjekte in Verbindung mit ihren Basismengen-Kriterien (siehe 4.2.1) exakt die Punkte obiger Anforderungsanalyse erfüllen. Was an dieser Stelle für die Architektur eines Gesamtsystems noch fehlt ist die Beschreibung eines Rahmens, in dem eine softwaretechnische Umsetzung der Layoutobjekte verwendet werden kann.

Sowohl die Grundlagen einer softwaretechnischen Umsetzung der Layoutobjekte als auch die Beschreibung des Rahmenkonzeptes für [ef:] folgen in den nächsten zwei Abschnitten.

7.3.1 Grundlagen der softwaretechnischen Umsetzung von Layoutobjekten

Layoutobjekt-Klassen in [ef:] sind Klassen im Sinne von objektorientierten Programmiersprachen, die von Programmierern realisiert werden müssen. Mit ihrer Hilfe werden die in Definition 28 eingeführten Layoutobjekte der Designspezifikationsmethodik [em:] softwaretechnisch umgesetzt. Ein Layoutobjekt in [ef:] hat somit im Gegensatz zu [em:] eine operative Semantik. Instanzen von Layoutobjekten werden in [ef:] als Komponenten zur Bildung von Designstrukturen verwendet.

Das Boxenmodell

Wie in Abschnitt 4.2.1 gezeigt wurde ist das Boxenmodell ein zur Berechnung des Layouts eines Dokumentes geeignetes Datenmodell, mit dem heute Dokumentformatierer arbeiten. Um ein Dokument zu formatieren, werden in diesem Modell die Bestandteile eines Dokumentes als Boxen (siehe Abbildung 2.21 auf Seite 51) betrachtet. Nun kann eine Box nicht nur atomar sein (und Beispielsweise einen Glyphen oder eine Graphik darstellen), sondern auch aus einer Menge von Boxen, die aufgrund einer Formatierfunktion angeordnet sind, bestehen; eine Eigenschaft, die beliebig tief verschachtelt gelten kann. Mit Hilfe dieser verschachtelten Boxen werden z.B. Formeln und Tabellen, aber insgesamt auch ganze Dokumente, realisiert.

In bisherigen Dokumentformatierern werden Boxen *von außen* mit Hilfe von im Dokumentformatierer monolithisch realisierten Formatierfunktionen positioniert. Es gibt einen mehr oder weniger zentralen Programmblock, der für die gesamte Anordnung beliebiger Boxen verantwortlich ist. Das bedeutet, dass sich diese eine Prozedur um die Bestimmung der Größen aller Boxen sowie deren Positionierung kümmert. Dies hat unmittelbar zur Folge, dass diese Prozedur alle verschiedenen Boxen gemäß verschiedener Formatierfunktionen verarbeiten können muss. Die Boxen sind also ohne Intelligenz, diese ist vollkommen monolithisch in den Kern des Dokumentformatierers verlagert. Als weitere Konsequenz des zentralen Programmblockes sind alle Informationen, die im Rahmen des Formatierprozesses anfallen, *global* verfügbar und auch änderbar.

Der Nachteil dieser Art von Dokumentformatierern wird deutlich, wenn ihnen neue Funktionalität zugefügt werden soll: Dazu sind unter Umständen große Teile des Programms zu verstehen und zu überarbeiten. Erschwert wird diese Situation zusätzlich durch die global in Dokumentformatierern änderbare Information, deren beliebige Verwendung nur schwer feststellbare Nebeneffekte bewirken kann. Bei zunehmender Komplexität eines Dokumentformatierers wird die Arbeitsweise des zentralen Programmblockes, in dem alle Formatierfunktionen miteinander verwoben sind, immer komplexer, damit immer schwerer nachvollziehbar und deshalb letzten Endes auch immer schwerer zu warten bzw. zu ergänzen.

Layoutobjekte: Ein erweitertes Boxenmodell

Im Modell von [ef:] hingegen werden die Layoutobjekte genannten Boxen um *eigenständige Intelligenz* erweitert. Layoutobjekte werden zwar nach wie vor von außen positioniert, sie können aber aufgrund ihrer eigenen operativen Semantik ihnen untergeordnete Bestandteile selbst aktiv anordnen. Mit diesem Ansatz wird das monolithische Konzept bestehender Dokumentformatierer aufgebrochen, die sämtliche Funktionalität in einem Block realisiert haben. Statt dessen wird die Implementierung von Funktionalität gemäß einzelnen Basisaufgaben auf die einzelnen Layoutobjekt-Klassen verlagert.

Bei dieser Verlagerung von Funktionalität weg von einem zentralen Programmblock in die Layoutobjekte wird nach dem Prinzip verfahren, dass die einzelnen Layoutobjekt-Klassen nur gerade soviele Aufgaben wie nur eben nötig übernehmen (siehe hierzu auch 4.2.2). Dieses Verfahren hat den Vorteil, dass der zu realisierende Dokumentformatierer leicht wartbar und

erweiterbar wird. Hierzu müssen lediglich einzelne Klassen geändert oder hinzugefügt werden. Auch ist der entstehende Code durch dieses Vorgehen weitgehend redundanzfrei. Weiterhin greifen Instanzen der einzelnen Klassen nicht mit den allgemein bekannten Nachteilen auf einen global manipulierbaren Informationsbestand zu, sondern halten eigenverantwortlich lokal ihre benötigten Informationen oder kommunizieren gemäß objektorientierten Paradigmen mit anderen Layoutobjekten über definierte Schnittstellen. Instanzen von Layoutobjekt-Klassen können aufgrund dieses Vorgehens beliebig mit Instanzen von anderen Layoutobjekt-Klassen zusammenarbeiten.

Als Einschränkung hierzu sei erwähnt, dass es gewisse Layoutobjekt-Klassen gibt, für die hier gewisse Abstriche vorgenommen werden müssen. Als Beispiel sei ein Fußnoten Layoutobjekt genannt, welches selbstverständlich einem Seitenumbruchs Layoutobjekt explizit für seine korrekte Bearbeitung bekannt sein muss. Dieser Aufbruch des *Anonymitätsprinzips*, wonach beliebige Layoutobjekt-Klassen miteinander arbeiten können, liegt jedoch in der Aufgabenstellung begründet (ein Seitenumbruch muss Fußnoten zu deren besonderer Behandlung kennen) und stellt somit kein konzeptionelles Problem dar.

Klassenhierarchie

Die Layoutobjekt-Klassen in [ef:] sind alle von einer Basisklasse `LayoutObject` abgeleitet. Weiterhin werden folgende vier Zwischenklassen eingeführt, von denen dann die einzelnen konkreten Klassen, die im Dokumentformatierer und Betrachter [ef:] zur Verfügung gestellt werden sollen, abzuleiten sind:

Zielobjekt (`DisplayObject`) Zwischenklasse für alle Layoutobjekte, die ein physikalisches Medium (z.B. Papier oder Bildschirm) darstellen.

Anordner (`ArrangerObject`) Zwischenklasse für alle Layoutobjekte, die Nachfolger aktiv anordnen.

Quellobjekt (`SourceObject`) Zwischenklasse für alle Layoutobjekte, die initial Boxen produzieren (z.B. für Glyphen oder Graphiken).

Modifikator (`ModifierObject`) Zwischenklasse für alle Layoutobjekte, die das Erscheinungsbild von Boxen verändern können, ohne allerdings deren Anordnung zu beeinflussen.

Eine Übersicht über alle verfügbaren Layoutobjekte und ihre Funktionalität ist in Anhang A zu finden. Die Klassenhierarchie der Layoutobjekte ist in Abbildung 7.1 dargestellt.

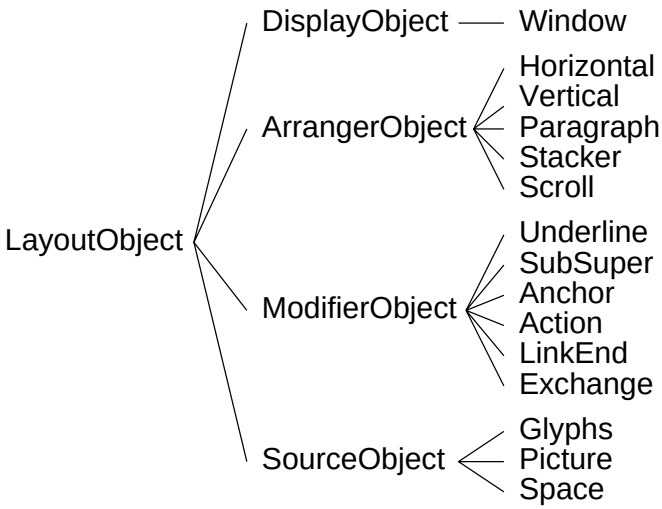

Abbildung 7.1: Klassenhierarchie der Layoutobjekte

7.3.2 Das Framework

Im letzten Abschnitt wurden die Grundlagen für die softwaretechnische Umsetzung der Layoutobjekte beschrieben. Dabei wurde ausführlich auf das Boxenmodell und die verteilte, modularisierte Realisierung der Formatierfunktionen sowie die Tatsache, dass sich die in einer Designstruktur enthaltenen Layoutobjekte gegenseitig anordnen, eingegangen. Um diese gegenseitige Anordnung auszuführen sowie für weitere Aktionen ist es natürlich nötig, dass die Layoutobjekte miteinander interagieren.

Das Layoutobjekt-Interface

Damit diese Interaktion zwischen den Layoutobjekten funktioniert, ist es notwendig, mit Hilfe des hier vorgestellten Frameworks ein gemeinsames Interface für die Layoutobjekte zu definieren. Bei der Verwendung einer objektorientierten Programmiersprache ist dieses Interface Bestandteil der Basisklasse, wodurch alle weiteren direkt oder indirekt abgeleiteten Layoutobjekt-Klassen dieses Interface als abstrakte Basis automatisch vorgegeben bekommen.

Dieses Interface besteht neben weiteren Hilfsmethoden, auf die hier nicht näher eingegangen werden soll, insbesondere aus den drei Methoden `computeBoxesAndPositions`, `draw` sowie `notify`. Die Methode `computeBoxesAndPositions` implementiert die spezifische Formatierfunktion, die durch das jeweilige Layoutobjekt realisiert wird. Als Resultat der Ausführung dieser Methode kennt ein Layoutobjekt die Dimensionen seiner Box sowie die relativen Positionen aller seiner Kind-Layoutobjekte zu sich selbst. Die Methode `draw` dient zur Darstellung eines Layoutobjektes auf einem Zielmedium. Mit Hilfe der Methode `notify` wird die Interaktion der Nutzer mit im Betrachter angezeigten Layoutobjekten (z.B. dem `Scroll` Layoutobjekt) behandelt.

Das Framework Protokoll

Für das Framework fehlt nun noch ein zentraler Mechanismus, mit dessen Hilfe die drei Prozesse des Formatierens, Ausgebens und Interagierens, die in dem Formatierer und Betrachter [ef:] möglich sind, koordiniert und gesteuert werden können. Dieser Mechanismus wird in Form von drei unabhängigen Protokollen realisiert.

Das Protokoll für den *Formatierprozess* steuert die Positionierung und Dimensionierung aller in einer Designstruktur enthaltenen Layoutobjekte. Dieses Protokoll wird gemeinsam von dem Framework und den einzelnen Layoutobjekten realisiert: Das Framework stößt die Methode `computeBoxesAndPositions` bei dem Wurzelobjekt der Designstruktur an, welches die Methode entsprechend seiner Formatierfunktion rekursiv an seine einzelnen Kind-Layoutobjekte weitergibt. Das Protokoll für den Formatierprozess deckt damit in seiner Funktionalität den Aufgabenbereich eines eigenständigen Formatierers ab.

Das Protokoll für den *Ausgabeprozess* realisiert teilweise die Funktionalität eines Betrachters. Dabei wird erneut auch bei diesem Protokoll eine Aufgabenverteilung zwischen dem Framwork und den in einer Designstruktur enthaltenen Layoutobjekten durchgeführt: Das Framework stößt zunächst die Methode `draw` bei dem Wurzelobjekt der Designstruktur an; dieses Layoutobjekt reicht die Methode wiederum an seine Kind-Layoutobjekte weiter.

Das Protokoll für den *Interaktionsprozess* realisiert die restliche Funktionalität eines Betrachters, die von dem Protokoll des Ausgabeprozesses nicht abgedeckt wird: Die Verarbeitung von Nutzereingaben. Auch bei diesem Protokoll findet eine Aufgabenverteilung zwischen dem Framework und den Layoutobjekten statt.

Die Aus- und Eingabe

Die Aus- und Eingabe, die Layoutobjekte auf konkrete Ausgabegeräte vornehmen bzw. von Eingabegeräten bekommen, wird in [ef:] über ein abstraktes Gerätekonzept (siehe [27]) realisiert. Im Rahmen dieses abstrakten Gerätekonzeptes gibt es eine abstrakte Basisklasse, die verschiedene Methoden zur Manipulation (wie z.B. Linien- und Textdarstellung, Koordinatentransformationen, Clipping-Funktionen und Eingabeverwaltung) eines abstrakten Gerätes zur Verfügung stellt. Mit Hilfe dieser Methoden können in der `draw` Methode eines Layoutobjektes über von der abstrakten Geräteklasse abgeleitete konkrete Geräteklassen Ausgaben auf ein konkretes Gerät vorgenommen bzw. in der `notify` Methode Eingaben von einem bestimmten konkreten Gerät akzeptiert werden. Hierzu muss lediglich einmal an zentraler Stelle im Framework die Unterstützung für das gewünschte Gerät realisiert werden.

Dieses Gerätekonzept schirmt die Layoutobjekte von jeglichen Hardwareabhängigkeiten ab, wodurch alle implementierten Layoutobjekte in einer beliebigen Designstruktur für ein beliebiges Ausgabemedium verwendet werden können (z.B. ein `Scroll` Layoutobjekt in einem Seitenumbruch Layoutobjekt). Mit Hilfe des abstrakten Gerätekonzeptes wird [ef:] somit auf einfache und effiziente Weise zu einem Cross-Media Formatierer und Betrachter System.

7.4 Die Prozessabläufe in [ef:]

Die Arbeitsgrundlage des integrierten Dokumentformatierers und Betrachters [ef:] ist eine Designstruktur, die ein Baum von attributierten Layoutobjekten ist und als gegeben betrachtet wird. Die Aufgabe von [ef:] ist zunächst, die gegebene Designstruktur zu formatieren. Im Rahmen dieses Prozesses wird die Designstruktur in die sogenannte vollständige Designstruktur überführt, in der alle Layoutobjekte die Ausmaße für ihre eigene Box sowie die Positionen für alle ihre Kind-Layoutobjekte kennen. Im Anschluß an diese Formatierung ist das berechnete Ergebnis in Form einer Layoutstruktur je nach Zielobjekt entweder für die folgende Druckausgabe in eine Datei zu schreiben oder am Bildschirm darzustellen und durch die Nutzer manipulieren zu lassen. Zur Erfüllung dieser Aufgaben interagieren die in der Designstruktur enthaltenen Layoutobjekte im Wesentlichen über die bereits genannten drei Methoden `computeBoxesAndPositions`, `draw` und `notify` in den drei Prozessen der Formatierung, Ausgabe und Interaktion miteinander. Abbildung 7.2 stellt diesen Ablauf graphisch dar. Im Folgenden wird auf die drei eben genannten Prozesse ausführlich eingegangen.

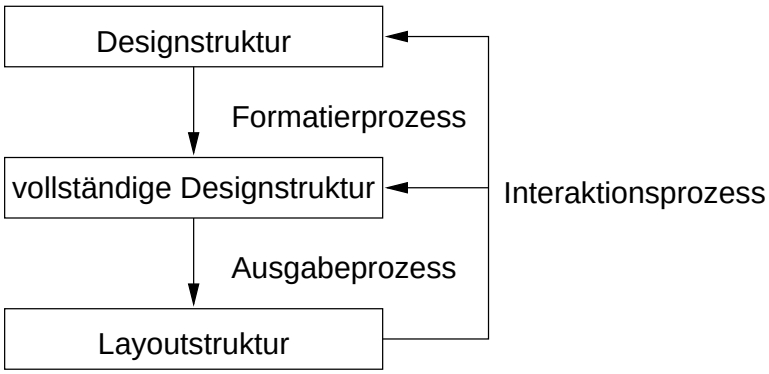

Abbildung 7.2: Die möglichen Prozesse in [ef:]

7.4.1 Der Formatierprozess

Der Formatierprozess in [ef:] dient der Steuerung der Funktionalität, die auch in einem konventionellen Dokumentformatierer zur Berechnung seiner Kernaufgaben anzutreffen ist (siehe hierzu auch Abschnitt 2.4). Zur Steuerung des Formatierprozesses ist im Framework von [ef:] ein Protokoll vorgegeben. Dort ist festgelegt, dass zum Start des Formatierprozesses das Framework bei dem Wurzelobjekt der Designstruktur die Methode `computeBoxesAndPositions` aufruft. Diese Methode ist bei jedem Layoutobjekt individuell zu realisieren und ist für die Berechnung der eigenen Boxausmaße sowie gegebenenfalls für die Positionierung aller Kind-Layoutobjekte gemäß der zu realisierenden Formatierfunktion verantwortlich.

Aufgrund der Aufgabenverteilung zwischen dem Framework und den Layoutobjekten übernehmen nach dem Anstoßen des Wurzelobjektes durch das Framework die Layoutobjekte selbst die Weitergabe des Methodenaufrufs `computeBoxesAndPositions`; dies ist der natürliche Weg, da Layoutobjekte in [ef:] selbst ihre Formatierfunktion realisieren und somit wissen,

wann exakt diese Methode an ihre Kind-Layoutobjekte geschickt werden muss. In der Regel gibt jedoch jedes Layoutobjekt die Methode `computeBoxesAndPositions` zunächst an alle seine Kinder weiter um sicherzustellen, dass alle diese Layoutobjekte ihre korrekten Abmessungen besitzen. Unter dieser Voraussetzung kann nun jedes Layoutobjekt die Anordnung seiner Kind-Layoutobjekte vornehmen.

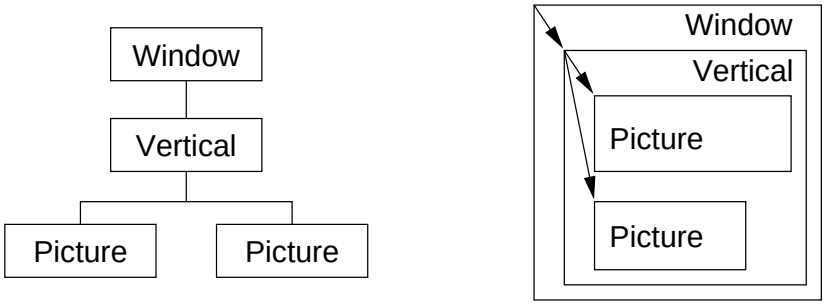

Abbildung 7.3: Beispiel eines Formatierprozesses: Die links gegebene Designstruktur berechnet die rechts dargestellte Anordnung von Layoutobjekten.

Abbildung 7.3 wird im folgenden Beispiel zur Veranschaulichung des prinzipiellen Formatierprozesses in [ef:] benutzt. Zunächst wird das `Window` Layoutobjekt vom Framework mit der Methode `computeBoxesAndPositions` aktiviert. Das `Window` Layoutobjekt reicht diesen Aufruf an das `Vertical` Layoutobjekt weiter, welches die Methode wiederum in Folge an die beiden `Picture` Layoutobjekte weitergibt. Da die `Picture` Layoutobjekte Quellobjekte sind, terminiert bei ihnen diese rekursive Weitergabe des Methodenaufrufs; sie können unmittelbar aus dem durch sie geladenen Bild ihre Größe berechnen. Nachdem beide `Picture` Layoutobjekte die Kontrolle an das `Vertical` Layoutobjekt zurückgegeben haben, kann nun auch das `Vertical` Layoutobjekt seine Ausmaße berechnen, nachdem es die beiden `Picture` Layoutobjekte gemäß der in ihm realisierten Anordnungsfunktion relativ zu sich positioniert hat. Zum Abschluß der Berechnungen dieses Formatierprozesses gibt nun das `Vertical` Layoutobjekt die Kontrolle zurück an das `Window` Layoutobjekt, welches das `Vertical` Layoutobjekt in seine linke obere Ecke positioniert.

Buchstaben im Formatierprozess

Buchstaben nehmen in normalen Dokumenten eine zentrale Sonderstellung ein: Sie stellen objektmäßig betrachtet mit Abstand den größten Anteil von Objekten an einem Dokument. Würden Buchstaben als einzelne Instanzen einer Layoutobjekt-Klasse in [ef:] umgesetzt werden, würde deshalb sehr schnell die Speichergrenze auch heutiger Rechensysteme gesprengt werden. Andererseits benötigt aber z.B. ein `Paragraph` Layoutobjekt den Zugriff auf jeden einzelnen Buchstaben, den es enthält, um ihn positionieren zu können.

Zur Lösung dieser soeben geschilderten Problemstellung wird in [ef:] ein zweistufiger Ansatz gewählt. Zunächst werden zusammenhängende Folgen von Buchstaben in einem Layoutobjekt der Klasse `Glyphs` zusammengefasst. Als Konsequenz daraus wird der Forderung nach Speichereffizienz entsprochen. Dann wird die funktionale Forderung nach dem Einzelzugriff

auf die Buchstaben gelöst, indem bei Aufruf der Methode `computeBoxesAndPositions` das `Glyphs` Layoutobjekt *aufgefaltet* wird: Es erzeugt für jeden komprimierten Buchstaben ein eigenes `Glyph` Layoutobjekt. Die Menge dieser `Glyph` Layoutobjekte ist dann anstatt des einen `Glyphs` Layoutobjektes für das Vorgängerobjekt in der Designstruktur während des Formatiervorganges sichtbar.

Damit der Zugriff des Vorgängerobjektes auf die einzelnen `Glyph` Layoutobjekte anstelle auf das eine `Glyphs` Layoutobjekt während der Formatierung erfolgt, wird von jedem Layoutobjekt aus ausschließlich mit Hilfe von Iteratorfunktionen auf seine Kindobjekte zugegriffen. Das `Glyphs` Layoutobjekt kann auf diese Weise seinem Vorgänger einen *virtuellen* Baum zur Verfügung stellen, der in dieser Form explizit nicht existiert: Der Vorgänger sieht nur die einzelnen `Glyph` Layoutobjekte, das `Glyphs` Layoutobjekt ist für ihn unsichtbar. Das Konzept des Aus- und Einfaltens eines `Glyphs` Layoutobjektes sowie des virtuellen Baumes ist in Abbildung 7.4 dargestellt.

Mit dem Ziel, dass immer nur eine kleine Anzahl von `Glyphs` Layoutobjekten aufgefaltet ist und somit der Speichervorteil erhalten bleibt, der durch das Einfalten von `Glyphs` Layoutobjekten erreicht wurde, muss am Ende des Formatierprozesses das `Glyphs` Layoutobjekt wieder *eingefaltet* werden. Dazu muss das `Glyphs` Layoutobjekt jedoch einen Hinweis von seinem Vorgängerobjekt erhalten, wann dies zu tun ist. Aus diesem Grund wird in [ef:] nach Abschluß der Formatieraufgabe durch ein Layoutobjekt an seine Kinder die Methode `ready` geschickt. Sie ermöglicht z.B. dem `Glyphs` Layoutobjekt seine Einfaltung vorzunehmen. Die Methode `ready` wird gemäß dieser Einsicht seiner Notwendigkeit prinzipiell von allen Layoutobjekten sobald wie möglich nach dem Ende der Ausführung ihrer Formatierfunktion an ihre Kind-Layoutobjekte geschickt. Die Methoden `computeBoxesAndPositions` und `ready` werden also verschachtelt während des Formatierprozesses angewendet. Somit sind immer nur Teilbäume der Designstruktur aufgefaltet.

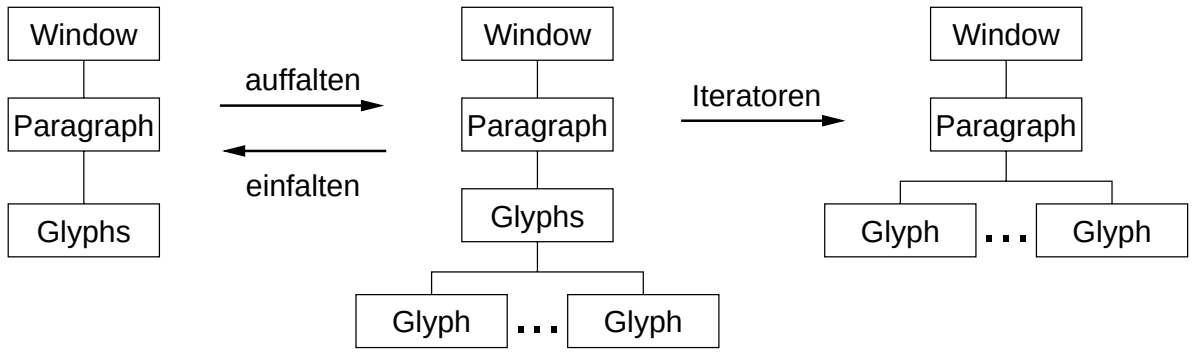

Abbildung 7.4: `Glyphs` Layoutobjekte im Formatierprozesses: Aus Speicherplatzgründen werden Sequenzen von Buchstaben zu einem `Glyphs` Layoutobjekt zusammengefasst. Zur Ausführung des Formatierprozesses muss das `Paragraph` Layoutobjekt jedoch Zugriff auf alle einzelnen Buchstaben haben, die es anordnen soll. Um dies zu realisieren wird das `Glyphs` Layoutobjekt in einzelne `Glyph` Layoutobjekte aufgefaltet und der Zugriff auf sie über einen Iterator gewährleistet. Am Ende des Formatierprozesses werden `Glyphs` Layoutobjekte wieder eingefaltet.

Beim Einfalten des `Glyphs` Layoutobjektes müssen die Positionen, an denen die einzelnen `Glyph` Layoutobjekte vom Paragraph Layoutobjekt positioniert wurden, vom `Glyphs` Layoutobjekt protokolliert werden. Sie werden für einen späteren Ausgabeprozess benötigt. Dies macht es nötig, dass jedes anordnende Layoutobjekt mit Hilfe der Methode `setPosition` den von ihm angeordneten Layoutobjekten ihre relative Position zu sich mitteilt. Sie kann dann unter anderem von dem `Glyphs` Layoutobjekt beim Einfalten ausgewertet werden.

Modifikatoren im Formatierprozess

Auch die Modifikator Layoutobjekte wie z.B. das `Underline` Layoutobjekt stellen für den Formatierprozess ein Problem dar. Modifikatoren werden zwischen anordnenden und den anzuordnenden Layoutobjekten eingesetzt. Es stellt sich die Frage, wie der Formatierprozess in einer derartigen Situation (siehe auch Abbildung 7.5 links) ausgeführt werden soll: Die Modifikator Layoutobjekte unterbinden den benötigten direkten Zugriff der anordnenden Layoutobjekte auf die anzuordnenden Layoutobjekte.

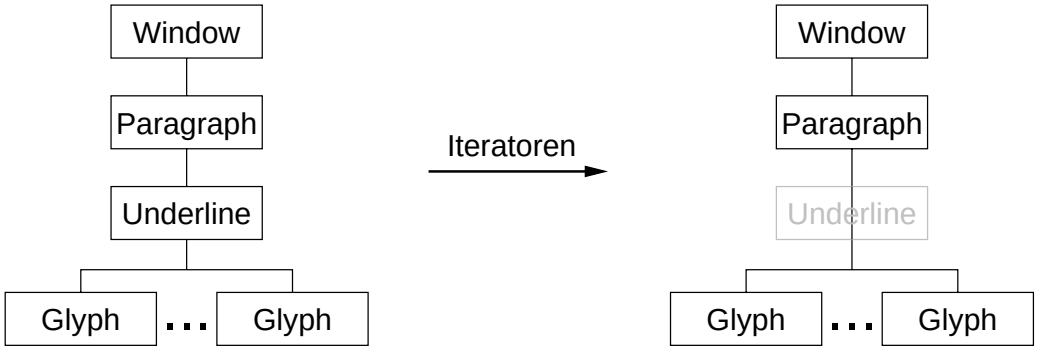

Abbildung 7.5: Modifikator Layoutobjekte im Formatierprozesses: Zum ungehinderten Zugriff auf die anzuordnenden Layoutobjekte machen sich Modifikator Layoutobjekte mit Hilfe der von ihnen realisierten Iteratoren durchsichtig.

Es ist offensichtlich, dass bei der linken Konstellation in Abbildung 7.5 das `Paragraph` Layoutobjekt die einzelnen `Glyph` Layoutobjekte positionieren soll, während das `Underline` Layoutobjekt für das `Paragraph` Layoutobjekt bei Ausführung der Formatierfunktion *durchsichtig* sein soll. Bei genauer Betrachtung dieser Konstellation fällt ihre Ähnlichkeit mit der Konstellation des störenden `Glyphs` Layoutobjektes aus dem vorherigen Abschnitt auf: Das `Underline` Layoutobjekt stört sein Vorgängerobjekt bei dem Zugriff auf seine Kinder. Somit liegt es nahe, auch hier das `Underline` Layoutobjekt *transluzent* zu machen. Dies wird mit Hilfe der Iteratorfunktionen, über die ein Layoutobjekt auf seine Kinder zugreift, erreicht. Modifikatoren geben bei einem Zugriff über einen Iterator nie sich selbst, sondern immer ihre Kinder als Ergebnis zurück; sie geben sich somit während des Formatierprozesses durchscheinend.

Modifikator Layoutobjekte nutzen die Methode `ready`, die ihnen am Ende des Formatierprozesses von ihrem Vorgängerobjekt geschickt wird, um Informationen zu berechnen, die sie

für die durch sie verkörperte Funktionalität benötigen. Das `Underline` Layoutobjekt holt sich hier z.B. die Positionen aller seiner Kinder, um die Unterstreichungslinien, die es beim Ausgabeprozess darstellen muss, berechnen zu können.

7.4.2 Der Ausgabeprozess

Nachdem der eigentliche Formatierprozess abgeschlossen ist, gilt es noch die dabei berechnete vollständige Designstruktur den Nutzern zur Darstellung zu bringen. Dies erfolgt in [ef:], indem die vollständige Designstruktur mit Hilfe der `draw` Methode, die für jedes Layoutobjekt gemäß der gewünschten Funktionalität zu implementieren ist, in eine Layoutstruktur überführt wird.

Zielobjekte sind in [ef:] diejenigen Layoutobjekte, die mit Hilfe des Frameworks einen Ausgabeprozess anstoßen. Durch ihre Eigenschaft, ein physikalisches Medium zu verkörpern, stellen sie originär ein konkretes Gerät in Form eines abstrakten Gerätes für den Ausgabeprozess zur Verfügung. Die Verwendung von abstrakten Geräten schirmt Layoutobjekte von jeglichen Hardwareabhängigkeiten ab. Abstrakte Geräte werden den Layoutobjekten als Parameter der `draw` Methode übergeben, damit sie darin ihre Ausgaben tätigen können. Für jedes Layoutobjekt wird von seinem jeweiligen Vorgängerobjekt in dem abstrakten Gerät ein Cursor zur Markierung der aktuellen Darstellungsposition sowie ein Clipping-Bereich zur Beschränkung des Ausgabebereiches gesetzt. Jedes Layoutobjekt kann sich relativ zu diesem Cursor in dem für ihn gesetzten Clipping-Bereich mit Hilfe von üblichen Ausgabefunktionen (wie Text- und Linienausgabe) gemäß seiner Darstellungsaufgabe darstellen.

Anordnende Layoutobjekte werden in der `draw` Methode in der Regel keine Ausgabe vornehmen, sondern lediglich den Cursor verschieben, bevor sie die Darstellungskontrolle mit Hilfe der `draw` Methode an ihre Kinder weiterreichen. Über dieses Vorgehen werden die beim Formatierprozess berechneten Positionen für die Kindobjekte in der erstellten Ausgabe realisiert. Modifikator Layoutobjekte wie z.B. das `Underline` Layoutobjekt können ihre eigene Darstellung realisieren, bevor oder nachdem sie die `draw` Methode an ihre Kinder weiterreichen. Speziell das `Glyphs` Layoutobjekt hat in der `draw` Methode die Aufgabe, alle durch es repräsentierten `Glyph` Layoutobjekte zur Darstellung zu bringen. Dies stellt aufgrund der Tatsache, dass es die Positionen aller durch es repräsentierten `Glyph` Layoutobjekte vor dem Einfalten aufgesammelt hat, jedoch kein Problem dar (siehe 7.4.1).

In Abbildung 7.6 wird symbolisch ein Ausgabeprozess für die vollständige Designstruktur dargestellt, die aufgrund der Designstruktur in Abbildung 7.3 berechnet wird.

7.4.3 Der Interaktionsprozess

Mit Hilfe des Dokumentformatierers und Betrachters [ef:] können Designstrukturen verarbeitet werden, die *interaktive* Layoutobjekte wie z.B. das `Scroll` Layoutobjekt beinhalten (mit dessen Hilfe wie in vielen Dokumentbetrachtern Dokumentinhalte gescrollt werden können). Interaktive Layoutobjekt können sich bei Eingaben durch Nutzer wie im Fall des eben erwähnten Scroll Layoutobjektes lediglich auf die Layoutstruktur auswirken (nur die Anzeige der Designstruktur verändert sich durch die Interaktion), oder im Falle der Menge von Layoutobjekten, mit

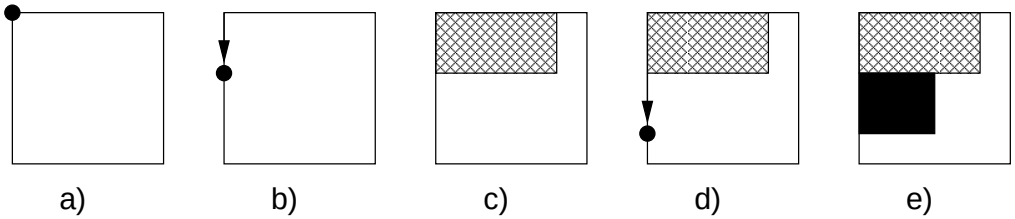

Abbildung 7.6: Beispiel eines Ausgabeprozesses: a) Zu Beginn des Ausgabeprozesses stellt das `Window` Layoutobjekt ein abstraktes Gerät zur Verfügung, setzt den Clipping-Bereich entsprechend seiner Größe und den Cursor in die linke obere Ecke. b) Dieses abstrakte Gerät wird an das `Vertical` Layoutobjekt weitergereicht, das den Cursor für die Ausgabe des ersten `Picture` Layoutobjektes positioniert. c) Das erste `Picture` Layoutobjekt stellt sich in dem ihm übergebenen abstrakten Gerät dar. d) Nachdem das erste `Picture` Layoutobjekt die Kontrolle wieder an das `Vertical` Layoutobjekt übergeben hat, kann dieses den Cursor für das zweite `Picture` Layoutobjekt positionieren. e) Das zweite `Picture` Layoutobjekt hat sich in dem abstrakten Gerät dargestellt.

denen Links in [ef:] realisiert werden, sogar Auswirkungen auf die Designstruktur haben. Zur einheitlichen Verarbeitung von Nutzereingaben gilt es somit für [ef:] ein Interaktionsprotokoll für den Interaktionsprozess zu entwickeln, mit dessen Hilfe Benutzereingaben an interaktive Layoutobjekte zugewiesen werden können.

Als Beispiel zur Erklärung des Interaktionsprotokolls wird die in Abbildung 7.7 dargestellte Designstruktur verwendet, in der innerhalb eines `Scroll` Layoutobjektes ein zweites `Scroll` Layoutobjekt angeordnet ist.

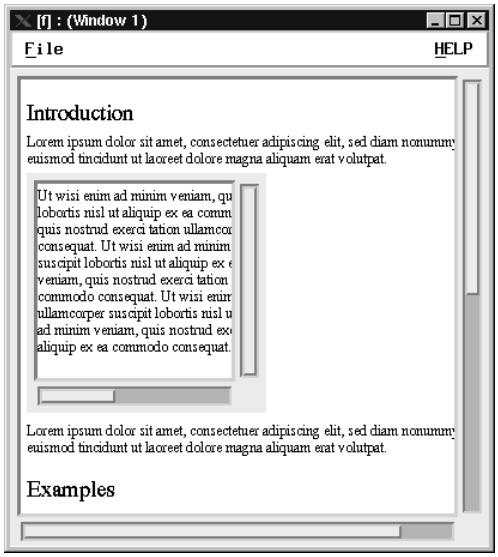

Abbildung 7.7: Interaktionsprozess in zwei vernesteten `Scroll` Layoutobjekten

Da in [ef:] ein `Scroll` Layoutobjekt nicht nur über die Scrollbalken manipuliert werden kann, sondern auch, indem bei gedrückter mittlerer Maustaste im inneren Bereich des `Scroll`

Layoutobjektes die Maus bewegt wird, ist folgende Frage naheliegend: Welches `Scroll` Layoutobjekt bekommt die Eingabe zugeordnet, wenn im inneren `Scroll` Layoutobjekt im mittleren Bereich bei gedrückter mittlerer Maustaste die Maus bewegt wird? Das Äußere, weil es immer alle Eingaben innerhalb seines inneren Bereiches erhält, oder das Innere, weil die Nutzerinteraktion doch *näher* an ihm, als an dem äußeren `Scroll` Layoutobjekt stattgefunden hat.

Das für [ef:] gewählte Protokoll legt für diese Frage mit Hilfe einer einfachen Regel eine eindeutige Entscheidung fest: Das Framework schickt bei einer Benutzerinteraktion an jenes der betroffenen interaktiven Layoutobjekte die Methode `notify` zur Information über eine Nutzerinteraktion, das von allen *betroffenen* interaktiven Layoutobjekten in der Designstruktur am tiefsten angeordnet ist. Kann dieses Layoutobjekt nichts mit der Nutzereingabe anfangen oder sie nur teilweise verarbeiten, so kann dieses Layoutobjekt die Methode `notify` in der Designstruktur hierarchisch nach oben an das nächste interaktive Layoutobjekt weiterreichen. Dieses Verfahren der Weitergabe kann sich bis zum letzen interaktiven Layoutobjekt in Richtung der Wurzel der Designstruktur fortsetzen.

Diese Form des Interaktionsprotokolls von [ef:] wurde aufgrund der Beobachtung gewählt, dass es exakt der Benutzererwartung entspricht, das tiefstgelegene interaktive Layoutobjekt über eine Nutzerinteraktion zu informieren: In der graphischen Darstellung ist dies in der Regel dasjenige Layoutobjekt, das am nächsten an einer Nutzerinteraktion lokalisiert ist. Somit ist die Antwort zu obiger Fragestellung, welches `Scroll` Layoutobjekt informiert wird: Das Innere.

Die Berechnung, welches Layoutobjekt das tiefste betroffene interaktive Layoutobjekt ist, wird in [ef:] über das abstrakte Gerätekonzept gelöst. Genauso wie es abstrakte Ausgabegeräte gibt, gibt es auch ein abstraktes Eingabegerät. Bei einer Nutzerinteraktion wird diesem zunächst die Position der Nutzereingabe mitgeteilt. Anschließend wird dieses abstrakte Eingabegerät als Parameter für einen Aufruf der `draw` Methode des Wurzelobjekts der Designstruktur verwendet. Das abstrakte Eingabegerät kann nun durch die normalen Ausgabevorgänge, die in den `draw` Methoden der Layoutobjekt vorgenommen werden, selbstständig berechnen, welches der betroffenen Layoutobjekte das der Nutzereingabe am nähestenliegende ist. Somit ist es nicht nötig, für Layoutobjekte spezielle Methoden zu realisieren, die die Berechnung übernehmen, ob sie von einer Eingabe betroffen sind: Die bereits implementierte `draw` Methode löst diese Aufgabe.

7.5 Umsetzung von Verweisstrukturen

Das einzige bisher im Interaktionsprozess beschriebene Layoutobjekt war das `Scroll` Layoutobjekt. Mit seiner Hilfe kann durch Interaktion eines Nutzers mit der Maus die Layoutstruktur und somit ausschließlich die Darstellung einer gegebenen Designstruktur verändert werden. Ein wichtiger Punkt, der heute insbesondere im WWW anzutreffen ist und noch nicht behandelt wurde, ist aber auch die Manipulation der Designstruktur selbst. Der Effekt dieser Manipulation der Designstruktur ist weitreichender als bei der bloßen Manipulation der Layoutstruktur: In diesem Fall wird nicht nur die Darstellung einer Designstruktur geändert, sondern auch der Inhalt der Designstruktur.

7.5.1 Grundlagen

Die Änderung der Designstruktur ist beispielsweise mit Hilfe der verfügbaren Browser für das WWW über sogenannte *Links* möglich: Klickt man auf einen Link, so wird z.B. das Fenster, in dem sich der angeklickte Link befindet, durch einen beliebig vorgebbaren neuen Inhalt ersetzt. Mit Hilfe von Links kann den Nutzern von [ef:] die Verweisstruktur (siehe Definition 9) eines Dokumentes zur Navigation durch das Dokument zugänglich gemacht werden. Die Links können in diesem Fall als interne Links bezeichnet werden; sie setzen Verweise von einem Dokument in dasselbe Dokument um. Neben diesen internen Links sind aber auch externe Links möglich; sie führen die Nutzer eines Dokumentes aus einem Dokument heraus in ein neues Dokument.

Verweise in Dokumenten werden also durch Links in den Benutzungsoberflächen von Betrachtern umgesetzt. Links bestehen dabei grundsätzlich zunächst aus einem sogenannten *Anker* (dem Ausgangspunkt für einen Link), mit dessen Hilfe die Nutzer eines Systems *aktiv* einen Link nutzen können. Zum anderen besitzen Links auch ein *Ziel*, auf das sie verweisen.

Durch die Interaktion mit einem Anker wollen die Nutzer eines Dokumentes Zugriff auf das zu dem Anker gehörige Ziel erhalten, das System soll für die Nutzer eine Aktion ausführen. Es stellt sich somit zum einen die Frage, in welcher Form ein Anker in der Benutzungsoberfläche von [ef:] den Nutzern zur Verfügung gestellt werden soll. Zum anderen ist zu klären, welche Aktionen Nutzer bei der Interaktion mit einem Anker auslösen können.

Die zur Verfügungstellung (graphische Darstellung und Manipulierbarkeit) eines Ankers für die Nutzer in der Benutzungsoberfläche von [ef:] kann (für eine Nutzung mit der Maus) zumindest über vier verschiedene Mechanismen erreicht werden. Möglich ist dies zunächst durch die *typographische Gestaltung* der Layoutbestandteile, die den Anker bilden. Dies kann z.B. durch Farbgebung oder Unterstreichung zur Auszeichnung der betroffenen Layoutbestandteile erfolgen. Weiterhin kann die Gestalt des Mauszeigers, wenn er in den Wirkbereich eines Ankers kommt, mit einem speziellen Symbol dargestellt werden. Schließlich kann auch in einer Statuszeile eine Meldung angezeigt werden, dass an der aktuellen Mausposition ein aktiver Anker ist. Die vierte Möglichkeit, die hier aber nicht weiterverfolgt werden soll, ist die zur Verfügungstellung von Links durch die verschiedenen Menüarten, die den Nutzern in einer Benutzungsoberfläche zur Verfügung gestellt werden können.

Allen oben genannten Möglichkeiten ist gemein, dass zu ihrer Nutzung Konventionen zwischen den Nutzern eines Dokumentes und den Erstellern der Dokumente getroffen werden müssen. Verdeutlicht wird dies aktuell besonders durch die Möglichkeiten, die in den neuen WWW-Browsergenerationen den Informationserstellern gegeben wird: Sie können textuelle Anker im Gegensatz zu früher graphisch beliebig gestalten – sogar derart, dass selbst geübte Nutzer Anker nicht mehr als Anker erkennen.

An dieser Stelle ist noch die Frage zu klären, welche Aktionen Nutzer bei Interaktion mit einem Anker auslösen können. Technisch betrachtet ist dies die Frage nach der prozeduralen Semantik eines Links, die auch mit Link-Semantik bezeichnet wird. Die sich in [ef:] ergebenden Möglichkeiten beruhen zunächst unmittelbar auf den graphischen Möglichkeiten, die im Rahmen einer zweidimensionalen Darstellung eines Dokumentes möglich sind. Dies ist zuerst

das direkte Ersetzen des Ankers durch das Linkziel (*Replacement Link* oder auch *Stretchtext* genannt), wobei das Linkziel in den weiterhin bestehenden Dokument-Kontext des Ankers einkopiert wird. Weiterhin kann der gesamte Inhalt des Fensters, in dem sich der Anker eines aktivierten Links befindet, durch das Linkziel ersetzt werden (auch *Goto Link* genannt). Anstelle der Ersetzung kann auch ein neues Fenster geöffnet werden, in dem das Linkziel dargestellt wird (auch *Parallel Link* genannt). Als weitere Möglichkeit bietet es sich noch an, das Linkziel an einer beliebigen bezeichneten Stelle in der graphischen Darstellung einzufügen, die in einem beliebigen bereits existierenden Fenster liegen kann.

Weiterhin können bei der Manipulation eines Ankers aber auch Aktionen ausgelöst werden, die nicht unmittelbar mit der graphischen Darstellung eines Dokumentes in [ef:] zusammenhängen, sondern z.B. weitere Programme starten. Ebenso sind neben den bisher aufgeführten möglichen einzelnen Aktionen natürlich auch Kombinationen denkbar, bei denen aufgrund eines Ankers mehrere Aktionen parallel ausgelöst werden. In diesem Fall spricht man auch von einem *Fat Link*.

7.5.2 Technische Umsetzung

Die technische Umsetzung des oben beschriebenen Konzeptes der Links wird in [ef:] mit Hilfe der vier Layoutobjekte `Anchor`, `Action`, `Linkend` und `Exchange` realisiert.

- **Das `Anchor` Layoutobjekt:**
  Das `Anchor` Layoutobjekt ist das Objekt, mit dessen Hilfe Anker in der Benutzungsoberfläche von [ef:] für den Interaktionsprozess realisiert werden. Soll ein gewisser Inhalt in der Layoutstruktur eines Dokumentes als Anker fungieren, so muß dieser Inhalt in der zugehörigen Designstruktur in einem `Anchor` Layoutobjekt als Vorgänger enthalten sein. Somit bilden also alle Layoutobjekte, die Nachfolger eines `Anchor` Layoutobjektes sind, in [ef:] einen interaktiven Bereich. Damit `Anchor` Layoutobjekte während des Formatierprozesses nicht störend wirken sind sie von der Klasse Modifikator abgeleitet – sie sind somit während des Formatierprozesses transluzent.

  Berührt ein Nutzer in der Benutzungsoberfläche von [ef:] mit der Maus einen von einem `Anchor` Layoutobjekt abgedeckten Bereich, so wird zur Kennzeichnung dieser Tatsache zum einen der Mauszeiger in Form einer Hand dargestellt. Dabei gilt bei verschachtelten `Anchor` Layoutobjekten wiederum das beim Interaktionsprozess eingeführte Konzept, dass der in der Designstruktur am tiefsten liegende betroffene `Anchor` als von der Interaktion zunächst betroffene ausgewählt wird. Neben dieser Mauszeigeränderung ist es weiterhin möglich, dass in einer Statuszeile ein Text ausgegeben werden kann. Dies wird ebenso wie die Aktionen, die bei einem Anklicken eines `Anchors` angegeben werden können, über das `Action` Layoutobjekt gesteuert.

- **Das `Action` Layoutobjekt:**
  Das `Action` Layoutobjekt kann Aktionen für einen Link definieren die dann ausgeführt werden, wenn ein `Anchor` Layoutobjekt mit der Maus berührt oder angeklickt wird. `Action` Layoutobjekte wirken bei verschachtelten `Anchor` Layoutobjekten immer nur für

das sie unmittelbar umgebende `Anchor` Layoutobjekt. Für jedes `Anchor` Layoutobjekt können beliebig viele `Action` Layoutobjekte eingesetzt werden, die dadurch festgelegten Aktionen werden beim jeweiligen Auslöser in der Reihenfolge abgearbeitet, wie sie in der Designstruktur in einem Tiefendurchlauf angetroffen werden.

Momentan können die folgenden Aktionen mit einem Hilfe von Action Layoutobjekten für `Anchor` Layoutobjekte spezifiziert werden:

- Ausführung eines Systemkommandos.
- Das Laden einer neuen Designstruktur.
- Das Fenster, in dem sich der Anker befindet, schließen.
- Den Inhalt eines benannten `Exchange` Layoutobjektes ändern.
- Innerhalb eines `Scroll` Layoutobjektes an eine bestimmte Stelle scrollen (in Zusammenarbeit mit einem `LinkEnd` Layoutobjekt).

- **Das `LinkEnd` Layoutobjekt:**
  Mit Hilfe eines `LinkEnd` Layoutobjektes wird in einer Layoutstruktur ein bestimmter Punkt als Ziel einer Scroll-Aktion eines `Scroll` Layoutobjektes markiert. Ein derart markierter Punkt kann als Ziel eines Vorganges dienen, der innerhalb eines `Scroll` Layoutobjektes an eine bestimmte Stelle scrollt. Ausserhalb eines `Scroll` Layoutobjektes hat ein `LinkEnd` Layoutobjekt keine Funktionalität.

- **Das `Exchange` Layoutobjekt:**
  Ein Exchange Layoutobjekt dient zur Realisierung von Replacement Links. Bei einer entsprechenden Aktion kann der Inhalt eines `Exchange` Layoutobjektes durch eine neue Designstruktur ersetzt werden.

  Bei der Ersetzung des Inhalts eines `Exchange` Layoutobjektes wird es nötig, die Designstruktur erneut zu formatieren. Dies wird in [ef:] mit Hilfe der Methode `recompute-BoxesAndPositions` ausgeführt. Nach der Änderung des Inhalts eines `Exchange` Layoutobjektes sendet dieses die Methode `recomputeBoxesAndPositions` an seinen Vorgänger, u.s.w.. Erreicht diese Methode ein `Scroll` oder ein `Window` Layoutobjekt (beide Objekte sind für eine Neuberechnug terminierend, da sie nach Außen feste Dimensionen haben) wird diese Weiterreichung unterbrochen und statt dessen ein Ausfruf der Methode `computeBoxesAndPositions` nach unten geschickt, damit sich die von der Änderung des geänderten `Exchange` Layoutobjektes betroffenen Layoutobjekte neu formatieren.

7.6 Zusammenfassung

Mit Hilfe des in dieser Arbeit entwickelten Dokumentformatierers und Betrachters [ef:] wurde zunächst ein erstes System geschaffen, mit dessen Hilfe die Designstrukturen der [em:] Designspezifikationsmethodik verarbeitet werden können.

Durch die Schaffung eines Frameworks mit seinen klar definierten Prozessen der Formatierung, Anzeige und Interaktion, den in ihm definierten Schnittstellen und den darin verwendeten Layoutobjekten wurde weiterhin ein schlanke Architektur erzielt, die monolithischen Dokumentformatierern überlegen ist. Sie ist übersichtlicher, einfacher zu warten und bringt den Vorteil mit sich, dass beliebige Formatierfunktionen miteinander problemlos zur Erreichung beliebiger Layouteffekte zu kombinieren sind.

Durch die Kapselung von Daten und Funktionalität in den Layoutobjekten wird weiterhin das Auftreten von Nebeneffekten bei der Realisierung und Ausführung von Formatierfunktionen ausgeschlossen. Die Einführung des abstrakten Gerätekonzeptes bewirkt dabei weiterhin, dass die in [ef:] in Form von Layoutobjekten realisierten Formatierfunktionen bei einmaliger Realisierung für beliebige Zielmedien verwendet werden können. Anforderungen der Speichereffizienz werden in [ef:] insbesondere durch die gewählte Realisierung der Glyphs Layoutobjekte Rechnung getragen.

Neue Formatierfunktionen, Medien und interaktive Komponenten können somit einfach in [ef:] in Form von neuen Layoutobjekten integriert werden. Die Vorteile, die sich bei der Realisierung von Layoutobjekten insbesondere bei der Beachtung der Basismengenkriterien für Layoutobjekte ergeben sind:

- Einfache Layoutobjekte sind einfach zu implementieren.
- Mehrfach benötigte Funktionalität wird nur einmal implementiert.
- Die Anzahl der benötigten Layoutobjekte ist klein.
- Bei Realisierung von [ef:] unbekannte Anforderungen können durch Kombination von mehreren Layoutobjekten realisiert werden.

Insgesamt ergibt sich somit mit [ef:] ein System, das aufgrund seiner Modularität auch zur Weiterentwicklung in Forschung und Lehre verwendet werden kann.

Kapitel 8

Schlußfolgerung und Ausblick

In diesem Kapitel werden abschließend in kompakter Form die Ergebnisse der vorliegenden Arbeit zusammengefaßt. Es wird aufgezeigt, welche Ziele mit dieser Arbeit erreicht wurden und welchen Beitrag diese Arbeit zur wissenschaftlichen Forschung leisten kann. Abschließend wird noch ein Ausblick auf mögliche weiterführende Arbeiten gegeben, die im Zusammenhang mit den in dieser Arbeit realisierten Konzepten und der dafür entwickelten Software erfolgen können.

8.1 Zusammenfassung

Ausgangspunkt für diese Arbeit war die unbefriedigende Situation, dass es im Bereich der rechnergestützten Dokumentverarbeitung trotz der Unmenge an verfügbaren Software-Werkzeugen ein enormes Methodendefizit gibt: Die Erstellung von Dokumenten erfolgt heute im Wesentlichen wie schon zu Beginn der rechnergestützten Dokumentverarbeitung mit Hilfe des Paradigmas des graphischen Auszeichnens von Dokumenten. Als unmittelbare Konsequenz dieses Vorgehens werden Dokumente in der Regel erstellt, gedruckt, gelesen und – abgelegt. Die digitale Vorlage des Dokumentes wird nicht weiterverwendet, sie ist in den meisten Fällen ein Einwegprodukt. Moderne Anforderungen bedingen jedoch eine vielfache Wiederverwendung eines Dokumentes, die mit Hilfe von automatisch ausführbaren Prozessen erfolgen soll.

Zur Lösung dieses Missstandes wurde zunächst der Stand der Technik im Bereich der digitalen Dokumenterstellung betrachtet und eine klare, eindeutige Nomenklatur für diese Arbeit geschaffen. Im Zuge dieser Vorarbeit ergab sich unmittelbar der erste Ansatzpunkt zur Lösung obig angesprochener Problematik der Einwegdokumente: Der Einsatz von logisch ausgezeichneten Dokumenten zur Separation der Dokumente von jeglicher Information, die zu ihrer Verarbeitung benötigt wird. Offen blieb jedoch die Frage nach den Methoden, die zur Verarbeitung von logisch ausgezeichneten Dokumenten verwendet werden sollen. Exakt an dieser Stelle herrscht derzeit der größte Mangel an verfügbaren Methoden.

Im Anschluss an diese Vorarbeit wurde deshalb eine Analyse des bisherigen Publikationsprozesses durchgeführt, seine Unterstützung in bestehenden Systemen untersucht und entschieden, für welche Bereiche eine Methodenunterstützung sinnvoll und nötig ist: Die automatische Steuerung der Kern- und der erweiterten Aufgaben von Dokumentformatierern.

Mit dieser Motivation wurde im Folgenden die Designspezifikationsmethodik [em:] zur Erstellung von Designspezifikationen erarbeitet. Besondere Aufmerksamkeit wurde bei ihrer Entwicklung auf die Personen der Zielgruppe gelegt, die mit der erstellten Methodik arbeiten sollen: Graphikdesigner. Sie sind im Publikationsprozess für den Arbeitsbereich, den Designspezifikationen funktional abdecken, verantwortlich. Folglich ist die Designspezifikationsmethodik [em:] entsprechend dieser Anforderung entwickelt worden. Kernkonzepte hierbei waren die zur Designspezifikation entwickelten Layoutobjekte, die Designstruktur, in der sie kombiniert werden, sowie die Pipeline-Metapher, die für die Nutzer von [em:] ein anschauliches Modell zum einfachen Verständnis der Methodik zur Verfügung stellt.

Zur Ergänzung der Designspezifikationsmethodik [em:] wurde weiterhin die kontextabhängige Verarbeitung von Dokumenten untersucht. Als Ergebnis hiervon wurde das T-Kontextmodell zur einfachen Spezifikation der kontextabhängigen Verarbeitung von Dokumentbestandteilen entwickelt. Weiterhin wurde ein Modell zur graphischen Spezifikation von T-Kontexten vorgestellt.

Aufbauend auf die Designspezifikationsmethodik [em:] und das T-Kontextmodell wurde anschließend eine Software-Architektur vorgestellt, mit deren Hilfe ein nutzerfreundliches, graphisches System [es:] zur Bearbeitung von Designspezifikationen realisiert wurde. Dieses besteht zunächst aus dem Designeditor [de:], dessen Ansatz *Design by Example* vorgibt, dass Designspezifikationen mit Hilfe von Beispieldokumenten einer Klasse von Dokumenten erstellt werden. Die derart erstellten Designspezifikationen können dann *automatisch* auf alle anderen Dokumente dieser Klasse mit Hilfe des Designprozessors [pe:], der die Designspezifikationsmethodik [em:] implementiert, angewendet werden.

Abschließend wurde der Dokumentformatierer und Betrachter [ef:] vorgestellt. Er ist in der Lage, die mit Hilfe der Designspezifikationsmethodik [em:] erstellten Designstrukturen zu formatieren, anzuzeigen und interaktiv von Nutzern manipulieren zu lassen.

Das Ergebnis dieser Arbeit ist eine ausgereifte und vollständig implementierte Designspezifikationsmethodik in Form eines nutzerfreundlichen Systems. Mit seiner Hilfe können Designspezifikationen zur Verarbeitungssteuerung von logisch ausgezeichnenten Dokumenten im Rahmen eines interaktiv graphischen Prozesses erstellt, getestet und ge*debuggt* werden. Weiterhin kann das erstellte System aufgrund seiner Batch-Fähigkeit im produktiven Publikationsprozess zur automatischen, mengenmäßigen Erstellung von Publikationsprodukten zum Einsatz kommen.

Neu an dieser Arbeit ist die Mächtigkeit des Funktionsumfanges der hier vorgestellten Designspezifikationsmethodik in Verbindung mit ihrer einfachen Verwendbarkeit. Das methodische Vorgehen bei ihrer Entwicklung hat zu einem Ansatz geführt, der aufgrund seines modularen Aufbaus leicht verständlich, erlernbar, universell und an neue Anforderungen anpassbar ist. Die mit der Designspezifikationsmethodik [em:] arbeitenden Graphikdesigner müssen bei ihrer

Arbeit nicht mehr wie in anderen bestehenden Systemen Designspezifikationen *programmieren*. Mit Hilfe der Designspezifikationsmethodik [em:] werden Designspezifikationen vielmehr auf einer den Nutzern angepassten Ebene *spezifiziert*.

Wissenschaftlich von besonderer Bedeutung ist weiterhin die vorgeschlagene Architektur des Dokumentformatierers und Betrachters [ef:]. Die in ihr vorgenommene Definition eines Frameworks mit seinen klar festgelegten Schnittstellen und Prozessprotokollen stellt ein neuartiges Konzept für Dokumentformatierer und Betrachter dar, das durch die Aufsplittung und Modularisierung der Formatierfunktionen in absolut unabhängige Komponenten den bisherigen monolithischen Dokumentformatierern deutlich überlegen ist. Die für [ef:] gewählte Architektur eignet sich durch ihren Aufbau insbesondere für die weitere Verwendung in Forschung und Lehre: Neue Formatierfunktionen und beliebige im Rahmen eines Dokumentformatierers und Betrachters mögliche Konzepte können in ihr auf einfache Art und Weise realisiert und getestet werden.

8.2 Ausblick

Die Betrachtungen dieser Arbeit haben sich auf die grundlegende Entwicklung neuer Methoden zur Unterstützung der abschließenden Schritte (der Aufbereitung von Manuskripten) des digitalen Publikationsprozesses bezogen. Es galt hierbei vordringlich das in diesem Bereich bestehende Methodendefizit, das derzeit funktionale Auswirkungen auf die Praxis hat, zu überwinden. Ein Ziel, das im Rahmen dieser Arbeit unter funktionalen Gesichtspunkten auch erreicht wurde.

Neben den funktionalen Gesichtspunkten erfüllt die vorgestellte Lösung weiterhin sogar ergonomische Anforderungen: Sie ist deklarativ und an die Qualifikationen von Graphikdesignern, die keine spezielle Programmiererfahrung haben, angepasst.

8.2.1 Engere Erweiterungen

WYSIWYG-Aspekte

Das Setzen von Parametern für Layoutobjekte erfolgt in der bisherigen Realisierung des Designeditors [de:] jedoch über einen einfachen Attributeditor. Eine Tatsache, die nicht dem Vorgehen entspricht, das Graphikdesigner von ihrem bisherigen Arbeiten mit aktuell verfügbaren Dokumentverarbeitungssystemen gewöhnt sind. Sie arbeiten bisher meist mit Systemen, die das WYSIWYG-Paradigma unterstützen und ändern daher Verarbeitungsparameter üblicherweise mit Methoden der direkten Manipulation in einer WYSIWYG-Darstellung des zu bearbeitenden Dokumentes.

Ein analoges Vorgehen hierzu ist im Designeditor [de:] bisher nicht möglich. Die Darstellungskomponente [ef:], in der zur Parametersetzung nach dem WYSIWYG-Paradigma verarbeitete Dokumentbestandteile mit Methoden der direkten Manipulation bearbeitet werden

müssten, arbeitet derzeit nur unidirektional mit [de:] zusammen. Zur weiteren Professionalisierung des Systems könnte an dieser Stelle somit durch die Realisierung einer bidirektionalen Schnittstelle eine noch bessere Unterstützung der Graphikdesigner durch das System [es:] erreicht werden.

Modularisierung

Bisher ist es zwar im System [es:] möglich, eine einmal erstellte Designspezifikation auf eine ganze Klasse von Dokumenten anzuwenden und diese somit bei jeder weiteren Anwendung auf ein logisch ausgezeichnetes Dokument erneut zu verwenden. Die Skalierung dieser Wiederverwendung ist jedoch nur sehr grob.

Oft besteht im Gegensatz zu dieser eben vorgestellten groben Möglichkeit der Wiederverwendung einer ganzen Designspezifikation jedoch die Notwendigkeit, nur gewisse kleinere Teilaspekte von bestehenden Designspezifikationen in modularer Form wiederzuverwenden. Gemäß dem Vorschlag in [33] ist es deshalb naheliegend, im System [es:] insgesamt eine Modularisierung von Designspezifikationen vornehmen zu können. Diese würde sinnvollerweise auf der Ebene von logisch zusammengehörigen Verarbeitungsaufgaben stattfinden. Gemäß diesem Ansatz könnte z.B. die Zusammenfassung aller Designregeln zur Verarbeitung einer Tabelle vorgenommen werden und dieses Modul in verschiedenen Designspezifikationen verwendet werden.

Dieses Vorgehen der Modularisierung ist mit im Vergleich zum bisherigen Implementierungsaufwand des Systems verhältnismäßig wenig Aufwand im System [es:] zu realisieren und bringt mit einer Aktion die im Allgemeinen bekannten und anerkannten Vorteile der Modularisierung und Redundanzvermeidung mit sich.

Versionierung

Designspezifikationen unterliegen wie viele andere digitale Produkte einer ständigen Weiterentwicklung. Oft ist es jedoch aus unterschiedlichsten Gründen nötig, auf einen bestimmten Stand einer Designspezifikation zurückgreifen zu können, oder diesen von vornherein konsistent im Rahmen der Modularisierung in einer weiteren Designspezifikation zu verwenden.

Die Konzepte zu einer derartigen Versionierung von Designspezifikationen entfernen sich zwar von dem eigentlichen Kern dieser Arbeit, sie sind im Rahmen eines alltäglichen Praxiseinsatzes eines Systems zur Erstellung von Designspezifikationen aber nötig.

8.2.2 Umfassende Erweiterungen

Wie bereits erwähnt, hat sich diese Arbeit mit der grundlegenden Entwicklung neuer Methoden zur Unterstützung der *abschließenden* Schritte des Publikationsprozesses befaßt, um hier das bestehende Methodendefizit zu schliessen. Es fehlen jedoch auch im *vorderen* Bereich des Publikationsprozesses etablierte Methoden, die sich insbesondere mit der modularen Erstellung

und dynamischen Zusammensetzung von Dokumenten, deren Versionierung und Konsistenzerhaltung beschäftigen.

Die Dringlichkeit von Forschungstätigkeiten im *vorderen* Bereich der Dokumenterstellung ergibt sich unmittelbar aus der bereits in der Einleitung zu dieser Arbeit erwähnten enormen wirtschaftlichen Bedeutung, die Dokumente darstellen: 80% aller digital gespeicherten Informationen eines Unternehmens liegt in Form von Dokumenten vor. Aktuell ist dieses Wissen aber nicht in der modularen Form verfügbar, wie es nötig wäre – eine Einsicht, die in vielen Unternehmen (wie z.B. im Consulting- und Dokumentationsbereich) zwar schon angenommen ist, deren Lösung aber derzeit aufgrund fehlender Methoden nicht angegangen werden kann. Ohne diese Ergänzung stehen mit obiger Einsicht alle in dieser Arbeit erzielten Ergebnisse nur mit *halber* Wirksamkeit zur Verfügung.

Anhang A

Beschreibung der verfügbaren Spezifikationsobjekte in [es:]

Im Folgenden werden die in [es:] verfügbaren Spezifikationsobjekte gemäß der Gruppierung im Designeditor [de:] vorgestellt.

Die bei einigen Objekten einsetzbaren Farbwerte sind: `BLACK`, `BLUE`, `GREEN`, `RED` und `GREY`.

Die verfügbaren Fonts können wie folgt spezifiziert werden:
`TIMES`, `TIMES_ITALIC`, `TIMES_BOLD`, `TIMES_BOLD_ITALIC`,
`COURIER`, `COURIER_OBLIQUE`, `COURIER_BOLD`, `COURIER_BOLD_OBLIQUE`,
`HELVETICA`, `HELVETICA_OBLIQUE`, `HELVETICA_BOLD`,
`HELVETICA_BOLD_OBLIQUE`, `HELVETICA_NARROW`,
`HELVETICA_NARROW_OBLIQUE`, `HELVETICA_NARROW_BOLD`,
`HELVETICA_NARROW_BOLD_OBLIQUE`,
`LUCIDA_SANS`, `LUCIDA_SANS_OBLIQUE`, `LUCIDA_SANS_BOLD`,
`LUCIDA_SANS_BOLD_OBLIQUE`, `LUCIDA_SANS_TYPEWRITER` und
`LUCIDA_SANS_TYPEWRITER_BOLD`.

A.1 Layoutobjekte

A.1.1 Zielobjekte

Window

Das `Window` Objekt hat keinen Ausgang und stellt die es erreichenden Layoutbestandteile in einem Fenster auf dem Bildschirm dar. Das Fenster verfügt über eine Menüzeile (zum Schließen des Fensters und Drucken des Fensterinhalts) und optional über eine Infozeile, in der mit Hilfe des Anchor Objektes Texte angezeigt werden können. Die Attribute zur Fenstergeometrie sind vom Typ `number`, um dieselben Variablen wie bei anderen Objekten verwenden zu können.

Die Umrechnung in Pixelzahlen erfolgt derzeit über den festen Umrechnungsfaktor von einem Pixel pro Punkt (= 72 dpi).

Attribut	Typ	Bedeutung
`name`	`string`	Text, der in der Titelleiste des Fensters angezeigt wird.
`height`	`number`	Höhe des Fensters in Pixeln.
`width`	`number`	Breite des Fensters in Pixeln.
`xPos`	`number`	x-Position des Fensters auf dem Bildschirm.
`yPos`	`number`	y-Position des Fensters auf dem Bildschirm.
`hasInfobar`	`string`	Soll eine Infozeile im Fenster angezeigt werden (`YES` | `NO`).
`infobarPos`	`string`	Spezifiziert, wo die Infozeile angezeigt wird (`TOP`|`BOTTOM`).

Trash

Das `Trash` Objekt führt zwar wie alle anderen Objekte eine Auswertung seiner Kinder aus (und kann somit Seiteneffekte hervorrufen), wirkt aber wie ein Zielobjekt: D.h., die das `Trash`-Objekt erreichenden Layoutbestandteile werden nicht weitergegeben, sondern ganz einfach verworfen.

A.1.2 Transiente Objekte

Scroll

Das `Scroll` Objekt umgibt es durchfließende Layoutbestandteile mit einem rechteckigen Bereich, in dem sie durch Benutzerinteraktion verschoben werden können. Vorraussetzung für die Verschiebbarkeit ist natürlich die Online-Darstellung des Dokuments auf dem Bildschirm.

Attribut	Typ	Bedeutung
`direction`	`string`	Kann die Werte `X`, `Y` und `BOTH` annehmen und gibt an, in welche Richtung eine Verschiebung möglich ist.
`height`	`number`	Höhe des Scrollbereichs.
`width`	`number`	Breite des Scrollbereichs.
`name`	`string`	Name des Scrolls.

Vertical

Das `Vertical` Objekt ordnet seine Kinder vertikal in einer Spalte an.

Attribut	Typ	Bedeutung
`margin`	`number`	Größe des Randes, der an allen Außenseiten des Objektes selbst freigelassen wird.
`skip`	`number`	Abstand der einzelnen Layoutobjekte voneinander.
`hAlign`	`string`	Bestimmt die horizontale Ausrichtung aller angeordneten Elemente zueinander. Mögliche Werte sind `LEFT`, `CENTER` und `RIGHT`.
`setBaseline`	`string`	Legt fest, an welcher Stelle nach aussen hin das Objekt seine eigene Baseline setzt. Mögliche Werte sind `TOP`, `CENTER`, `BOTTOM`, `FIRST_BASELINE` und `LAST_BASELINE`, wobei die ersten drei Werte zur Baselineberechnung alle angeordneten Objekte berücksichtigen, die letzten zwei Werte jeweils nur das erste bzw. letzte Objekt.

Horizontal

Das `Horizontal` Objekt ordnet es durchfließende Layoutbestandteile in einer horizontalen Zeile an.

Attribut	Typ	Bedeutung
`margin`	`number`	Größe des Randes, der an allen Außenseiten des Objektes selbst freigelassen wird.
`skip`	`number`	Abstand der einzelnen Layoutobjekte voneinander.
`vAlign`	`string`	Legt fest, wie die angeordneten Elemente zueinander vertikal ausgerichtet werden. Mögliche Werte sind `TOP`, `CENTER`, `BOTTOM` und `BASELINE`.
`setBaseline`	`string`	Legt fest, an welcher Stelle nach aussen hin das Objekt seine eigene Baseline setzt. Mögliche Werte sind `TOP`, `CENTER`, `BOTTOM` und `BASELINE`.

Stack

Das `Stack` Objekt ordnet alle eingehenden Layoutbestandteile übereinander geschichtet an (in der Z-Ebene). Im Normalfall findet keine Verschiebung der Elemente statt, außer es wird mit Hilfe des `StElemOffs` Objekts ein Verschiebungsvektor spezifiziert.

Attribut	Typ	Bedeutung
`margin`	`number`	Größe des Randes, der an allen Außenseiten des Objektes freigelassen wird.
`setBaseline`	`string`	Legt fest, an welcher Stelle nach aussen hin das Objekt seine eigene Baseline setzt. Mögliche Werte sind `TOP`, `CENTER`, `BOTTOM` und `BASELINE`.

StElemOffs

Verschiebt die ihn durchfließenden Layoutbestandteile in X- und Y-Richtung. Es sind nur positive Werte erlaubt (nach rechts und unten). Dieses Objekt arbeitet nur mit dem direkt darüberliegenden `Stack` Objekt zusammen.

Attribut	Typ	Bedeutung
`shiftXforStacker`	`number`	Gibt die Anzahl Pixel an, um die in X-Richtung verschoben werden soll.
`shiftYforStacker`	`number`	Gibt die Anzahl Pixel an, um die in Y-Richtung verschoben werden soll.

Paragraph

Das `Paragraph` Objekt ordnet es durchfließende Layoutbestandteile in Form eines Absatzes an, d.h. die Anordnung erfolgt in mehreren Zeilen, die jeweils höchstens die Breite des Absatzes erreichen. Ein Zeilenumbruchalgorithmus legt die Aufteilung der Zeilen fest.

Attribut	Typ	Bedeutung
`align`	`string`	Kann die Werte `LEFT`, `RIGHT` und `CENTER` annehmen und gibt an, wie der Absatz ausgerichtet wird.
`baselineSkip`	`number`	Zeilenabstand.
`firstIndent`	`number`	Größe des Erstzeileneinzugs.
`noIndentLines`	`number`	Anzahl der Zeilen, auf die sich der Erstzeileneinzug erstreckt.
`optLines`	`string`	Optimiert die Ausgabegeschwindigkeit von Zeichen im Online-Fall. Mögliche Werte sind `YES` und `NO`.
`setBaseline`	`string`	Kann die Werte `TOP`, `CENTER`, `BOTTOM`, `FIRSTLINE` und `LASTLINE` annehmen und gibt die Position der Grundlinie an.
`width`	`number`	Breite des Absatzes.
`lastLineDepth`	`number`	Gibt an, welche Unterlänge für die letzte Zeile eines Paragraphen mindestens gesetzt wird.

Underline

Das `Underline` Objekt gibt die von seinen Kindern empfangenen Layouteinheiten in unterstrichener Form an seinem Ausgang aus.

Attribut	Typ	Bedeutung
`distFromBaseline`	`number`	Abstand der unterstreichenden Linie zur Grundlinie.
`lineColor`	`string`	Farbe der unterstreichenden Linie.
`lineWidth`	`lenght`	Dicke der unterstreichenden Linie.

SubSuper

Das `SubSuper`-Objekt ermöglicht eine Hoch- bzw. Tiefstellung von Objekten und erlaubt damit z.B. die Realisierung von Indizes und Exponenten in mathematischen Formeln oder Fußnotenzeichen.

Attribut	Typ	Bedeutung
`distFromBaseline`	`number`	Abstand des hoch/tiefgestellten Objekts zur Grundlinie.

Tagger

Der `Tagger` stellt dem ihn durchfließenden Layoutstrom ein anzugebendes Prefix voran und setzt an das Ende ein Postfix. Auf Wunsch können auch seine aktuellen Attribute mit ausgegeben werden (entweder im Prefix- oder Postfix-Teil oder auch in beiden; jeweils zwischen `xxxStart` und `xxxEnd`).

Einsatzgebiet für dieses Objekt ist die Konvertierung von logisch strukturierten Dokumenten in strukturierte Form, z.B. nach HTML.

Attribut	Typ	Bedeutung
`prefixStart`	`string`	Erster Teil Prefix.
`prefixEnd`	`string`	Letzter Teil Prefix.
`suffixStart`	`string`	Erster Teil Suffix.
`suffixEnd`	`string`	Letzter Teil Suffix.
`AttsInPrefix`	`number`	Attribute werden (=1) bzw. werden nicht (=0) im Prefix ausgegeben.
`AttsInSuffix`	`number`	Attribute werden (=1) bzw. werden nicht (=0) im Suffix ausgegeben.

A.1.3 Quellobjekte

Reader

Der `Reader` liest die Layoutbestandteile, die er *erzeugt*, aus einer anzugebenden Datei.

Attribut	Typ	Bedeutung
`inputFileName`	`string`	Name der Datei, die Layoutbestandteile enthält.

Space

Das `Space` Objekt hat keinen Eingang. Es dient zur Erzeugung eines horizontalen und/oder vertikalen Leerraums, wie er beispielsweise zwischen einer Kapitelüberschrift und dem Fließtext (vertikal) oder innerhalb einer Kapitelüberschrift zwischen der Numerierung und dem Inhalt (horizontal) gewünscht wird.

Attribut	Typ	Bedeutung
`height`	`lenght`	Höhe des vertikalen Leerraums.
`width`	`lenght`	Breite des horizontalen Leerraums.

Picture

Das `Picture`-Objekt liefert das Bild mit dem angegebenen Dateinamen. Es hat keinen Eingang. Einzig derzeit unterstütztes Graphikformat ist XPM. Bei Bereitstellung mehrerer Bilder können diese auch animiert (im Wechsel) dargestellt werden.

Attribut	Typ	Bedeutung
`name`	`string`	Dateiname des gewünschten Bildes. Im Falle einer Animation wird an diesen Namen beim Ladevorgang eine Nummer angehängt, die die Sequenznummer des geladenen Bildes darstellt (Start bei 0).
`doLoop`	`string`	Gibt an, ob bei Bereitstellung mehrerer Bilder diese animiert dargestellt werden sollen (`YES` | `NO`).
`delayTime`	`number`	Spezifiziert, wie lange die einzelnen Bilder im Falle einer Animation dargestellt werden sollen (in ms).
`numberOfFrames`	`number`	Gibt vor, wieviele Bilder in der Animation enthalten sind.

Glyph

Das `Glyph` Objekt erzeugt ein graphisches Zeichen für den Buchstaben, der in seinem `char`-Attribut spezifiziert wird. Es hat keinen Eingang.

Attribut	Typ	Bedeutung
`char`	`string`	Logisches Zeichen, das formatiert werden soll.
`color`	`string`	Farbe des graphischen Zeichens.
`font`	`string`	Schriftart.
`fontSize`	`number`	Schrifgröße.

Glyphs

Das `Glyphs` Objekt verwaltet sämtliche Daten und Informationen, die zur Darstellung einer Sequenz von Buchstaben benötigt werden.

Attribut	Typ	Bedeutung
`input`	`string`	Logische Zeichen, die formatiert werden sollen.
`color`	`string`	Farbe der graphischen Zeichen.
`font`	`string`	Zu verwendende Schriftart.
`fontSize`	`number`	Zu verwendende Schriftgröße.
`rawOutput`	`number`	Wenn 1, wird nur der Text von `input` in Form eines Strings ausgegeben (dies ist in Zusammenhang mit dem `Tagger` Objekt nötig).

A.2 Auswertungsobjekte

Inheritance

Dieses ist eines der beiden Objekte, welches den Wert einer globalen Variablen (aus der Sicht der nachfolgenden Layoutobjekte) während eines Auswertungslaufs verändern kann. Das `Inheritance` Objekt verfügt hierzu über eine anwenderdefinierbare Variablenliste, welche sich nach außen hin wie die Attributliste eines normalen Layoutobjektes präsentiert. Das heißt, man kann den Variablen einen Ausdruck zuordnen. Dieser wird wie beim normalen Layoutobjekt berechnet; anschließend ist jedoch für alle im Teilgraphen *unter* dem `Inheritance` Objekt liegenden Objekte unter dem betreffenden Variablennamen nur noch dieser neu berechnete Wert zugänglich. Die in einem `Inheritance` Objekt modifizierte Variable hat rechts von diesem Objekt dagegen wieder ihren alten Wert. Kurz gesagt: Ein `Inheritance` Objekt ändert den Wert einer globalen Variablen genau für den unter ihm liegenden Teilbaum.

Modifier

Dies ist das zweite Objekt, welches den Wert einer globalen Variablen verändern kann. Hierzu bietet es wie das `Inheritance` Objekt eine benutzerdefinierbare Liste von Variablen an, denen der Anwender einen Ausdruck der [em:]-Ausdruckssprache zuordnen kann. Der Unterschied zum `Inheritance` Objekt besteht darin, dass das `Modifier` Objekt den Wert einer globalen Variablen beim Verlassen seines Unterbaumes verändert. Diese Änderung gilt für den gesamten weiteren Auswertungsdurchlauf. Zu beachten ist weiter, dass die Auswertung des Attributausdrucks erst zum Zeitpunkt der Wertänderung stattfindet, also die aktuellen Variablenwerte aus dem Rückfluss durch das Objekt genommen werden.

Repeater

Ein `Repeater` liest aus einer Datei die Werte für bestimmte Variablen, die er dann wie ein `Inheritance` an seine Kinder vererbt. In der betreffenden Datei können mehrere Sätze von Variablenwerten stehen; der Unterbaum des `Repeater` wird dann so oft mit jeweils den neuen Variablenwerten ausgewertet, wie Datensätze in der Datei sind.

Attribut	Typ	Bedeutung
`inputFileName`	`string`	Name der Datensatzdatei.

Der Dateiaufbau der Datensatzdatei wird hier kurz beschrieben: Die erste Zeile dient der Festlegung der Struktur eines Datensatzes, der Angabe des jeweiligen Typs der betreffenden Spalte sowie der Spezifikation eines Default-Wertes.

```
<type>^<varname>^<default> | ... <CR>
```

mit:

```
<type>     s = string, n = number
<varname>  Name der Variablen, die mit dem Wert der
           betreffenden Spalte belegt werden soll.
<default>  Default Wert fuer den Fall, dass in einem
           Datensatz der Wert nicht gesetzt wird
           (Beispiel hierfür: "... || ...").
```

Der Delimiter „|“ trennt einzelne Spalten. Anschliessend folgen beliebig viele Datensätze der Form:

```
<value> | ... <CR>
```

Die Anzahl der Werte muss mit der Strukturdefinition übereinstimmen. Soll für einen Wert der dort angegebene Default-Wert verwendet werden, so bleibt die Spalte einfach leer und es folgt sofort der Delimiter.

Blocker

Der `Blocker` verhindert die Auswirkung von Seiteneffekten, die in seinem Unterbaum vorgenommen wurden, auf Objekte oberhalb oder rechts von ihm. Man kann ihn sich als ein `Inheritance` Objekt vorstellen, das alle im System definierten Variablen mit ihrem aktuellen Wert auf einen Stack legt und beim Verlassen diese wieder vom Stack nimmt.

SetLogAtt

Dieses Objekt erzeugt im darüberliegenden logischen (!) Objekt neue Attribute aus allen Attributen, die es selbst besitzt. Diese können wie bei einem `Inheritance` Objekt aus der Variablenliste hinzugefügt und mit einem Ausdruck belegt werden. Diese Werte sind dann in jedem späteren logischen Fluss wie gewöhnliche Attribute logischer Objekte vorhanden.

String

`String` Objekte besitzen eine Reihe von internen Eingängen, an die jeweils beliebige Formatierungen gehängt werden können. `String` Auswertungsobjekte werten sequentiell für jedes einzelne Zeichen, das in ihrem `input` Spezifikationsattribut enthalten ist, anhand von für die internen Formatierungen gegebenen Bedingungen aus, welche dieser internen Formatierungen in die Designstruktur einkopiert wird. Die Auswertung eines `String` Auswertungsobjektes erfolgt also analog zu einer mehrfachen Auswertung (gesteuert durch eine `while`-Schleife über alle Zeichen der `input` Spezifikationsvariable) einer in üblichen Programmiersprachen verfügbaren `if`-Kaskade.

Attribut	Typ	Bedeutung
`input`	`string`	Enthält die zu verarbeitenden logischen Zeichen.

Connector

Grundsätzlich werten Layoutobjekte nur bis zum Ende der Formatierung aus, in der sie enthalten sind. Um eine weitere Auswertung der tiefer liegenden logischen Objekte anzustoßen, muss als letztes Objekt in einer Formatierung ein `Connector` hängen. Dieser erfüllt zwei Aufgaben: Zum Einen wählt er den ab sofort weiter auszuwertenden Fluss, ermöglich somit also einen Flusswechsel. Zum Anderen kann über das Attribut `evalAlgorithm` entschieden werden, in welcher Reihenfolge die tiefer liegenden logischen Objekte ausgewertet werden. Im Moment sind dabei die Werte 0 (Auswertung von links nach rechts) und 1 (Auwertung von rechts nach links, *rückwärts*) implementiert.

Attribut	Typ	Bedeutung
`inFlow01`	`number`	Nummer des Layoutflusses, der akzeptiert werden soll.
`evalAlgorithm`	`number`	Legt den Auswertungsalgorithmus bzw. die Auswertungsreihenfolge fest `(0|1)`.

A.3 Spezialobjekte

A.3.1 Objekte für die Benutzerinteraktion

Anchor

Alle Layoutbestandteile, die durch ein `Anchor` Objekt fliessen, werden zu einem Anker, also zu einem Bereich, mit dem der Benutzer in irgendeiner Form interagieren kann. Dies kann z.B. das Überstreichen oder Klicken mit der Maus in diesen Bereich sein. Die Möglichkeiten hierbei werden durch `Action` Objekte festgelegt.

Action

Ein `Anchor` Objekt sammelt alle Aktionen, die in den untergordneten `Action`-Objekten definiert sind, auf. Ein `Action` Objekt spezifiziert was passieren soll, wenn ein Nutzer mit dem Ankerbereich interagiert. Denkbar ist hier das Öffnen eines Fensters mit einem neuen Dokument oder das Ausführen eines System-Kommandos.

Attribut	Typ	Bedeutung
`shellCommand`	`string`	Gibt das Kommando an, das ausgeführt werden soll (falls `type` = `sysCommand`).
`scrollToPostion`	`string`	Legt fest, an welche Stelle (festgelegt durch ein `LinkEnd`-Objekt) gescrollt werden soll (in Kombination mit `linkRangeType` und falls `type` = `scrollToPosition`).
`linkRangeType`	`string`	Spezifiziert, wo im Formatierer der Scroll-Befehl Wirkung hat. Mögliche Werte sind `SCROLL`, `WINDOW` und `VIEWER`.
`file`	`string`	Gibt die Datei an, die hinzugeladen werden soll (für `type` = `loadFile` und `type` = `exchange`).
`exchangeLOname`	`string`	Legt den Namen des `Exchange` Objekts fest, dessen Inhalt mit dem Inhalt der geladenen Datei ausgetauscht werden soll. (falls `type` = `exchange`).
`hideWindow`	`string`	Gibt an, ob das aktuelle Fenster geschlossen werden soll (falls `type` = `hideWindow`).
`doInfoDisplay`	`string`	Steuert, ob der Informationstext angezeigt wird (falls `type` = `showInfo`).
`displayedInfo`	`string`	Legt den Text fest, der im Informationsbereich des Fensters angezeigt werden soll (falls `type` = `showInfo`).

Attribut	Typ	Bedeutung
type	string	Mögliche Werte sind: sysCommand, scrollToPosition, loadFile, exchange, hideWindow und showInfo. Je nach Wert führt das Action Objekt einen Shell-Aufruf durch (gesteuert durch shellCommand), scrollt an eine bestimmte Position (gesteuert durch scrollToPosition und linkRangeType), lädt eine neue zusätzliche Datei in den Formatierer, tauscht den Inhalt eines Exchanger Objektes aus (gesteuert durch exchangeLOname), schließt das aktuelle Fenster oder zeigt eine Information im Informationsbereich (falls vorhanden) des aktuellen Fensters an (gesteuert durch displayedInfo und doInfoDisplay).

LinkEnd

Ein LinkEnd Objekt markiert eine bestimmte Stelle in einer Designstruktur. Alles, was darunter liegt, gehört zu diesem Zielpunkt eines Links. Die Markierung geschieht über eine zu vergebende ID.

Attribut	Typ	Bedeutung
name	string	Gibt den Namen (die ID) für dieses Link-Ende an.

Exchanger

Ein Exchanger tauscht auf Anforderung durch einen Anchor (welcher diese Information wiederum aus einer ihm zugeordneten Action Objekt bezieht) seinen Inhalt mit dem Inhalt einer spezifizierten Datei aus.

Attribut	Typ	Bedeutung
setBaseline	string	Legt fest, an welcher Stelle nach aussen hin das Objekt seine eigene Baseline setzt. Mögliche Werte sind TOP, CENTER, BOTTOM, FIRST_BASELINE und LAST_BASELINE, wobei die ersten drei Werte zur Baselineberechnung alle angeordneten Objekte berücksichtigt, die letzten zwei Werte jeweils nur das erste bzw. letzte Objekt.
name	string	Name des Exchange Objekts, um von einem Anchor aus angesprochen werden zu können.

A.3.2 Spezifikationshilfen

SysCall

Dieses Objekt übergibt das spezifizierte Kommando der Systemshell zur Ausführung.

Attribut	Typ	Bedeutung
`shellCommand`	`string`	Auszuführendes Shell-Kommando.

Protocol

Das `Protocol` Objekt protokolliert in einer anzugebenden Datei all das mit (d.h. die Layoutbestandteile), was durch dieses Objekt fließt.

Attribut	Typ	Bedeutung
`outputFileName`	`string`	Datei, in welche geschrieben werden soll. Diese kann z.B. von einem `Reader` Objekt gelesen werden.

Empty (auch Null-Layoutobjekt)

Dieses Objekt hat keine Attribute und dient nur als *Anker* für andere Objekte. Es läßt alle Layoutbestandteile ungehindert passieren.

A.3.3 Sonstige

Beamer

Das `Beamer` Objekt sammelt alle durch ihn fließenden Layoutbestandteile auf (soweit agiert es quasi wie ein `Protocol` Objekt), legt sie aber nicht in einer Datei, sondern intern ab, wobei es diese Objektansammlung mit einem anzugebenden Namen versieht.

Attribut	Typ	Bedeutung
`key`	`string`	Name bzw. ID, unter welchem die Objekte abgelegt werden.

Materializer

Ein `Materializer` Objekt fungiert wie ein `Reader` Objekt mit dem Unterschied, dass es nicht aus einer Datei liest, sondern die Objekte zur Verfügung stellt, die von einem `Beamer` Objekt vorher im System unter einem bestimmten Namen hinterlegt wurden.

Sind mehrere Objektansammlungen mit demselben Namen vorhanden, so werden sie alle in undefinierter Reihenfolge geliefert. Haben Attribute von enthaltenen Objekten die Option `Deferred Evaluation` gesetzt, so werden die Attribute an der Stelle, an der der `Materializer` wirkt, neu berechnet. Anderenfalls werden die Werte genommen, die zum Zeitpunkt des Aufsammelns durch das `Beamer` Objekt gültig waren.

Wurden die Objektansammlungen durch ein `SortingKey`-Objekt an ihrer Ursprungsstelle beim Aufsammeln durch das `Beamer` Objekt mit einem Key versehen, so kann das `Materializer` Objekt bei mehrfachem Vorhandensein einer Objektansammlung mit gleichem Namen die Ausgabe in aufsteigender oder absteigender Sortierung nach dem Key veranlassen.

Attribut	Typ	Bedeutung
`key`	`string`	Name bzw. ID der Objektansammlung.
`sortingOrder`	`number`	0=aufsteigend, 1=absteigend (jeweils lexikographisch).

SortingKey

Setzt im direkt darüber befindlichem `Beamer` Objekt den Key für eine spätere Sortierung der Ausgabe durch einen `Materializer`. Sind mehrere `SortingKey`-Objekte unterhalb eines einzigen `Beamer`s, so gilt immer der zuletzt erzeugte bzw. berechnete Key.

Attribut	Typ	Bedeutung
`sortingKey`	`string`	Schlüssel, mit welchem der `Beamer` die Objektansammlung versieht

Literaturverzeichnis

[1] Adobe Systems Incorporated. Portable document format reference manual. Addison-Wesley Publishing Company, 1993.

[2] Adobe Systems Incorporated. PostScript language reference manual third edition. Addison-Wesley Publishing Company, 1999.

[3] Wolfgang Appelt. Dokumentaustausch in Offenen Systemen: Einführung in die ISO-Norm 8613. Springer-Verlag, Berlin Heidelberg New York, 1990.

[4] Arbortext. URL: http://www.arbortext.com/

[5] Michael von Bressensdorf, Technischer Leiter beim Deutschen Taschenbuch Verlag. Persönliche Kommunikation, 1999.

[6] Thomas Bretthauer, Anne Brüggemann-Klein, Stefan Hermann and Rolf Klein. A Component Architecture for Cross-media Formatters. In Roger. D. Hersch, Jacques André and Heather Brown, editors. Electronic Publishing, Artistic Imaging, and Digital Typography. Seiten 444-453. Springer-Verlag, Berlin Heidelberg New York, 1998.

[7] Anne Brüggemann-Klein. Formal Models in Document Processing. Habilitationsschrift vorgelegt an der Mathematischen Fakultät der Albert-Ludwigs-Universität zu Freiburg i.Br., Freiburg, 1993.

[8] Anne Brüggemann-Klein. Einführung in die Dokumentenverarbeitung. B. G. Teubner Stuttgart, 1989.

[9] Anne Brüggemann-Klein. Vorlesung Elektronisches Publizieren.
URL: http://www11.informatik.tu-muenchen.de/lehre/lectures/ws1998-99/ep.html, 1998.

[10] Anne Brüggemann-Klein. Vorlesung Hypermedia.
URL: http://www11.informatik.tu-muenchen.de/lehre/lectures/ss1998/hm.html, 1998.

[11] Anne Brüggemann-Klein and Stefan Hermann. Design by Example: A user-centered approach to the specification of document layout. In F. Rowland and J. Meadows, editors. Electronic Publishing '97: New Models and Opportunities. Proceedings of an ICCC/IFIP Conference held at the University of Kent at Canterbury, England, 14-16 April 1997. Seiten 223-236. ICCC Press, 1997.

[12] Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie, Bonn. Richtlinien für die Teilnahme am Gründerwettbewerb Multimedia 1998.
URL: http://www.gruenderwettbewerb.de, 1998.

[13] James Clark. Jade – James' DSSSL Engine. URL: http://www.jclark.com/jade/, 1999.

[14] James Clark. nsgmls. URL: http://www.jclark.com/sp/nsgmls.htm, 1999.

[15] W. Clinget, J. Rees, editors. Revised Report on the Algortithmic Language Scheme.
URL: ftp://cs.indiana.edu/pub/scheme-repository/doc/standards/r4rs.ps.gz, 1996.

[16] Peter Coad, Edward Yourdon. Objektorientierte Analyse, 2. Auflage. Prentice Hall Verlag, München, 1996.

[17] Dave Collins. Designing Object-Oriented User Interfaces. Benjamin/Cummings Publishing Company, Redwood City, CA, 1995.

[18] J. H. Coombs, A. H. Renear, S. J. DeRose. Markup systems and the future of scholarly text processing. Communications of the ACM 30, no. 11, Seiten 933-947, 1987.

[19] Tristan Daude. Unterstützung verteilter nichtlinearer Videonachbearbeitung durch CSCW. Herbert Utz Verlag, München, 1998.

[20] DPA. Meldung 200 923, 1995.

[21] Dudenverlag. Duden Deutsches Universalwörterbuch. Dudenverlag, Mannheim/Wien/Zürich, 1989.

[22] R. Furuta, J. Scofield, A. Shaw. Document Formatting Systems: Survey, Concepts, and Issues. Computing Surveys, 14(3): 417-472, September 1982.

[23] C. F. Goldfarb. The SGML Handbook. Clarendon Press, Oxford, 1990.

[24] Rudolf Paulus Gorbach, Lehrbeauftragter für Typographie an der FH München. Persönliche Kommunikation, 1999.

[25] V. N. Gudivadu. Multimedia Systems - An Interdisciplinary Perspective, ACM Comp. Surveys, Vol. 27, No. 4, Seiten 545-548, Dec. 1995.

[26] Jürgen Gulbins, Christine Kahrmann. Mut zur Typographie: Ein Kurs für DTP und Textverarbeitung. Springer-Verlag, Berlin Heidelberg New York, 1993.

[27] Stefan Hermann. Diplomarbeit am Institut für Informatik: Entwurf und Implementierung eines objektorientierten Oberflächenkonzeptes für ein Programmsystem zur Erfassung und Auswertung von bei der Gensequenzierung anfallenden Daten unter besonderer Berücksichtigung von in der Forschung entstehenden Sicherheitsbedürfnissen und softwareergonomischen Aspekten. TU München, 1994.

[28] ISO. ISO 2382-23. Information Technology - Vocabulary - Part 23: Text Processing.
International Organization for Standardization, 1994.

[29] ISO. ISO 8613. Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format. International Organization for Standardization, 1989.

[30] ISO. ISO/IEC 10179. Information technology - Processing languages - Document Style Semantics and Specification Language (DSSSL). International Organization for Standardization, 1996.

[31] ISO. ISO 13522-1. Information technology - Coding of multimedia and hypermedia information - Part 1: MHEG object representation. International Organization for Standardization, 1997.

[32] ISO. ISO 8879. Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML). International Organization for Standardization, 1986.

[33] J. Johnson, R.J. Beach. Styles in document editing systems. IEEE Computer, Seiten 32-43, January 1988.

[34] Donald. E. Knuth. TeX and METAFONT: New Directions in Typesetting. American Mathematical Society and Digital Press, Bedford, Massachusetts, 1979.

[35] Donald. E. Knuth. The TeXbook. Addison-Wesley, Reading, MA, 1986.

[36] Helmut Kopka. LaTex: Einführung Band 1. Addison-Wesley, 1994.

[37] Marshall Lee. Bookmaking: The illustrated guide to design/production/editing. R. R. Bowker Company, New York, 1979.

[38] H. W. Lie. Cascading HTML style sheets - a proposal.
URL: http://www.w3.org/People/howcome/p/cascade.html, 1994.

[39] Meyer. Meyers Enzyklopädisches Lexikon, Band 15. Meyer, Mannheim, 1975.

[40] Microsoft Technical Support. RTF Version 1.5. Microsoft, 1997.

[41] Mozilla.org. Our mission. URL: http://www.mozilla.org/mission.html, 1999.

[42] Ethan V. Munson, Charles Nicholas, Derick Wood (Eds.). Principles of Digital Document Processing. Springer-Verlag, Berlin Heidelberg New York, 1998.

[43] Makoto Murata and Koichi Hayashi. Formatter Hierarchy for Structured Documents. In C. Vanoirbeek and G. Coray, editors. *EP92, The Cambridge Series on Electronic Publishing*. Seiten 77-94. Cambridge University Press, Cambridge, 1992.

[44] Jakob Nielsen. Hypertext & Hypermedia. Academic Press, Inc., San Diego, CA, 1990.

[45] Steve Pepper. The Whirlwind Guide to SGML & XML.
URL: http://www.infotek.no/sgmltool/products.htm, 1999.

[46] Andy Reinhard. Managing the New Document. Byte, Seiten 90-104, August 1994.

[47] Wolfgang Rieger. SGML für die Praxis. Springer-Verlag, Berlin Heidelberg New York, 1995.

[48] U. Riehm, K. Böhle, I. Gabel-Becket, B. Wingert. Elektronisches Publizieren: eine kritische Bestandsaufnahme. Springer-Verlag, Berlin Heidelberg New York, 1992.

[49] Richard Sietmann. Zirkelspiele – Die wissenschaftliche Literaturversorgung steckt weltweit in der Krise. c't magazin für computer technik. Heise Verlag, Ausgabe 20, 1999.

[50] Ralf Steinmetz. Multimedia-Technologie: Einführung und Grundlagen. Springer-Verlag, Berlin Heidelberg New York, 1993.

[51] University of Chicago Press. The Chicago Manual of Style. The University of Chicago Press, Chicago, 1982.

[52] Xinxin Wang and Derick Wood. A Conceptual Model for Tables. In Ethan V. Munson, Charles Nicholas and Derick Wood, editors. Principles of Digital Document Processing. Seiten 10-23. Springer-Verlag, Berlin Heidelberg New York, 1998.

[53] W3C. Cascading Style Sheets, level 1.
http://www.w3.org/TR/1999/REC-CSS1-19990111, 1999.

[54] W3C. Cascading Style Sheets, level 2.
URL: http://www.w3.org/TR/1998/REC-CSS2-19980512, 1998.

[55] W3C. CSS3 module: W3C selectors.
URL: http://www.w3.org/TR/1999/WD-CSS3-selectors-19990803, 1999.

[56] W3C. Extensible Markup Language (XML) 1.0.
URL: http://www.w3.org/TR/1998/REC-xml-19980210.ps, 1998.

[57] W3C. Extensible Stylesheet Language (XSL) Specification.
URL: http://www.w3.org/TR/1999/WD-xsl-19990421, 1999.

[58] W3C. What are style sheets? URL: http://www.w3.org/Styles, 1999.

Index