# Entwurf und Implementierung einer Hypertext–Komponente für die LIGA–Prozessmodellierung

**Studienarbeit**
der
Fakultät für Informatik,
Universität Fridericiana zu Karlsruhe,

durchgeführt am
Institut für Angewandte Informatik,
Kernforschungszentrum Karlsruhe GmbH

von

cand. inform. **Torsten Neck**

**März 1994**

Referent: **Professor Dr. H. Trauboth**
Institut für Angewandte Informatik
Kernforschungszentrum Karlsruhe GmbH

Betreuer: **Dr. E. Holler**
Institut für Angewandte Informatik
Kernforschungszentrum Karlsruhe GmbH

# 0 Verzeichnisse

## 0.1 Inhalt

## 0.2 Abbildungen

# 1 Aufgabenstellung

## 1.1 Umfeld

Am Institut für Angewandte Informatik des Kernforschungszentrums Karlsruhe wurde eine wissensbasierte Software zur Unterstützung der Fertigung von Mikrostrukturen nach dem LIGA–Verfahren (Röntgentiefenlithografie und Galvanoplastik) entwickelt. In ihr werden die Fertigungsprozesse in einzelnen Prozeßschritten modelliert. Dabei werden jedem einzelnen Prozeßschritt kennzeichnende Attribute und Methoden zugewiesen, die Abfolge von Prozeßschritten in einem Fertigungsteilprozeß oder Fertigungsprozeß wird durch entsprechende Vernetzung der Prozeßschritte modelliert.

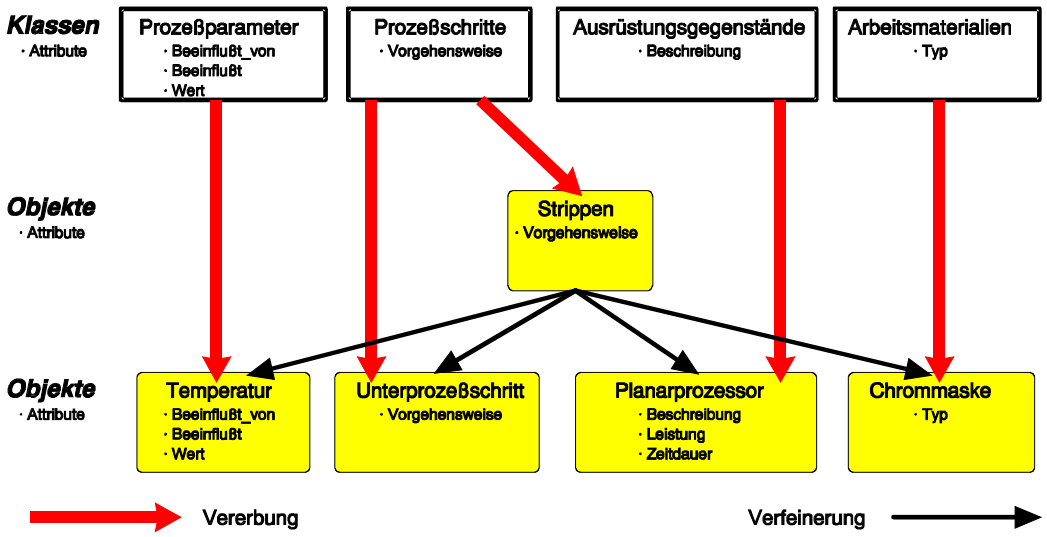


*Abb. 1: Objektorientierte Modellierung des LIGA–Fertigungsverfahrens in /Brauch92/*

Bei der Modellierung des einzelnen Prozeßschrittes kommen objektorientierte Techniken zur Anwendung: jeder reale Prozeßschritt ist *Instanz* einer *Klasse* „Prozeßschritte" und kann somit Attribute und Methoden dieser Klasse *erben*. Darüber hinaus sind dem Prozeßschritt weitere Instanzen untergeordnet, die zur Beschreibung des Verfahrens zusätzlich benötigte Klassen repräsentieren. So können für jeden Schritt zugehörige Ausrüstung, verwendete Rohstoffe und Materialien und Prozeßparameter beschrieben werden. Abbildung 1 zeigt die eben beschriebene Struktur (s. /Brauch92/).

Neben der rechnergestützten Haltung von Wissen über Fertigungsschritte liegt eine große Zahl unterschiedlicher, schriftlicher Informationen vor. Diese schriftlich fixierten Informationen reichen von Handbüchern und Bedienungsanleitungen für die bei der Fertigung eingesetzten Geräte über interne und externe Publikationen zu ausgewählten

Aspekten der Mikrostrukturtechnik und Mikrostrukturfertigung oder zusammenfassende Darstellungen bis hin zu Notizen der Experten und Operateure, die Erfahrungen und Rezepte bei der Arbeit mit bestimmten Apparaturen festhalten. Sie sind insbesondere für eine Einbringung in die Prozeßmodellierung als Attribute einzelner Prozeßschritte nicht geeignet.

## 1.2 Eigentliche Aufgabenstellung

Die Aufgabe dieser Studienarbeit ist es, ein Konzept zu entwickeln, das es ermöglicht, in die bestehende Modellierung mit geeigneten Werkzeugen auch das zuvor genannte informelle Material zu integrieren.

Zunächst bedeutet dies, daß eine geeignete Technik bereitzustellen ist, um auf das Schrifttum am Bildschirm zugreifen zu können. Darüber hinaus soll das vorliegende Material jedoch auch derart „zugänglich" gemacht werden können, daß Zusammenhänge zwischen den Informationen transparent werden und eine enge Kopplung an die bereits informationstechnisch modellierten Prozeßschritte möglich ist.

Die Kopplung an das bestehende Prozeßmodell darf jedoch nicht den ausschließlichen Zugang zum vorhandenen, informellen Material bilden, eine Sichtung des gesamten Dokumentbestandes muß unabhängig davon von jeder beliebigen Workstation des zugrundeliegenden Rechnernetzes möglich sein.

## 1.3 Ansatz

Diese Arbeit setzt zur Lösung der gestellten Aufgabe mit einer Hypertext–Komponente an: den LIGA–Konstrukteuren soll als Werkzeug zur bestehenden, informationstechnischen Prozeßmodellierung ein einfach zu bedienendes Hypertextsystem an die Hand gegeben werden. Es ermöglicht eine gezielte Sichtung des zuvor beschriebenen, informellen Materials während des Entwurfsprozesses der LIGA–Fertigung auch aus der informationstechnischen Prozeßmodellierung heraus. Zum inhaltlichen Aufbau des Hypertextsystems, der aufgrund erforderlichen Fachwissens von den Experten aus der Mikrostrukturtechnik weitgehend selbst vorzunehmen ist, muß ein weiteres, einfach zu bedienendes Werkzeug bereitgestellt werden, das das Generieren der später beschriebenen Netzstruktur eines Hypertextsystems ermöglicht.

Das dabei durch bestehende Implementierungen vorgegebene Rechnerumfeld besteht aus UNIX–Workstations der *„Sparc"*–Serie des Herstellers *Sun Microsystems Inc.*, die unter dem UNIX–SVR4[1]–Derivat *„Sun OS 4.1.2"* vernetzt betrieben werden.

[1] SVR4 ist die geläufige Abkürzung für „System V, Release 4", den derzeitigen De–facto–Standard in den verschiedenen UNIX–Implementierungen.

Die Applikationen basieren auf der grafischen Benutzeroberfläche *X–Windows* (kurz *X* oder *X11*) des *Massachusetts Institute of Technology (MIT),* beziehungsweise auf dem von Sun Microsystems bislang ausgelieferten *Network Windows System (NeWS). X11* ist ein portables, netzwerkfähiges Windows–System für Grafikbildschirme, das auf vielen Rechnerarchitekturen läuft und viele verschiedene Grafikoptionen auf der Basis von Bitmaps unterstützt. Das proprietäre *NeWS* bietet in etwa die gleiche Funktionalität wie *X11*, baut dabei jedoch gänzlich auf die portable Seitenbeschreibungssprache *PostScript* der *Adobe Systems Inc.* auf.

Das zu entwickelnde Hypertextkonzept muß sich in dieses Hard– und Softwareszenario eingliedern, muß aber andererseits ermöglichen, die unterschiedlichen Dokumentquellen auch auf anderen Architekturen zu erschließen. Aus diesem Grunde wurde *PostScript* als verbindliches Dokumentformat gewählt. Es läßt sich auf allen relevanten Architekturen (Workstations, Personal Computer, XEROX–Schreibsysteme) generieren und kann auch aus dem Papiermedium mit Hilfe von Scannern erzeugt werden.

In den nachfolgenden, einführenden Überlegungen zu Hypertext wird streng zwischen dem *Hypertextautor* und dem *Hypertextleser* unterschieden. Im Kontext dieser Arbeit ist es sinnvoll, auch die Rollen *(konventioneller) Autor* und *Hypertextredakteur,* als besondere Ausprägung des Hypertextautors, zu unterscheiden: Die vorliegenden Dokumente sind zumeist von den Autoren als herkömmliche Texte verfaßt und ursprünglich nicht für die Vernetzung in einem Hypertextsystem vorgesehen. In einem weiteren Schritt muß daher ein Hypertextredakteur das bestehende Material derart aufarbeiten, daß er zunächst Verbindungen und Verweise von Textteilen und Abbildungen in den Dokumenten und zwischen verschiedenen Dokumenten konzipiert, dann jedoch auch eine Zuordnung von modellierten Prozeßschritten zu Hypertextdokumenten herstellt. Diese Zuordnungen liefern zusätzliche Artikulationspunkte neben den sich aus der vorgegebenen Gliederung in Dokumente natürlich ergebenden Eintrittsstellen in das entstehende Netz. Der Hypertextredakteur muß dazu insbesondere über Fachwissen aus dem Gebiet verfügen, aus dem die redaktionell aufzubereitenden Dokumente stammen, sein datenverarbeiterisch technisches Fachwissen kann sich auf die Kenntnis der im Abschnitt 2.2 beschriebenen Link–Technik (konkretisiert auf die Eigenschaften das tatsächlich eingesetzten Systems) beschränken. Im Sinne eines ökonomischen Arbeitens nach dem Pipelineprinzip, kann die tatsächliche Einarbeitung der Links in die Dokumentdaten von einer weiteren Instanz, die nun kein Fachwissen über das behandelte Gebiet mehr benötigt, durchgeführt werden.

Unabhängig von der zuvor beschriebenen, produktiven Seite des Systems ist die konsumierende zu sehen. Dem *Hypertextleser* ist ein einfach zu handhabendes Werkzeug zur Verfügung zu stellen, das ihm den Zugang in das Hypertext–Netz von den verschiedenen Eintrittspunkten aus ermöglicht, von ihm jedoch keine Kenntnis über die Topologie des Netzes erfordert. Er muß lediglich das Prinzip der Links kennen, diese im realen System erkennen können und die Interaktionsmechanismen zu ihrer Verfolgung beherrschen.

Die Abgrenzung der verschiedenen Rollen beeinflußt auch das zu entwickelnde Konzept; es sind zwei selbständige Komponenten bereitzustellen:

- ❏ eine *Navigationskomponente* für den Hypertextleser, die das Anzeigen beliebiger PostScript–Programme ermöglicht und neben dem konventionellen Blättern die Verfolgung und Rückverfolgung von Links bietet,

und unabhängig von ihr

- ❏ ein *Hypertexteditor* für den Hypertextredakteur oder die erfassende Person, der — ohne den Navigationsbetrieb im bestehenden Netz signifikant zu stören — die Veränderung der Netzstruktur erlaubt.

Es ist darüber hinaus noch denkbar, dem Hypertextautor für seine redaktionelle Tätigkeit Werkzeuge bereitzustellen, die eben diese Tätigkeit unterstützen. Entsprechende Konfektionssoftware für diesen Zweck — im wesentlichen Datenhaltung — ist sicherlich erhältlich. Die Gewinnung von Informationen über ein bereits bestehendes Hypertext–Netz ist über Anpassung bekannter Algorithmen aus der Graphentheorie möglich.

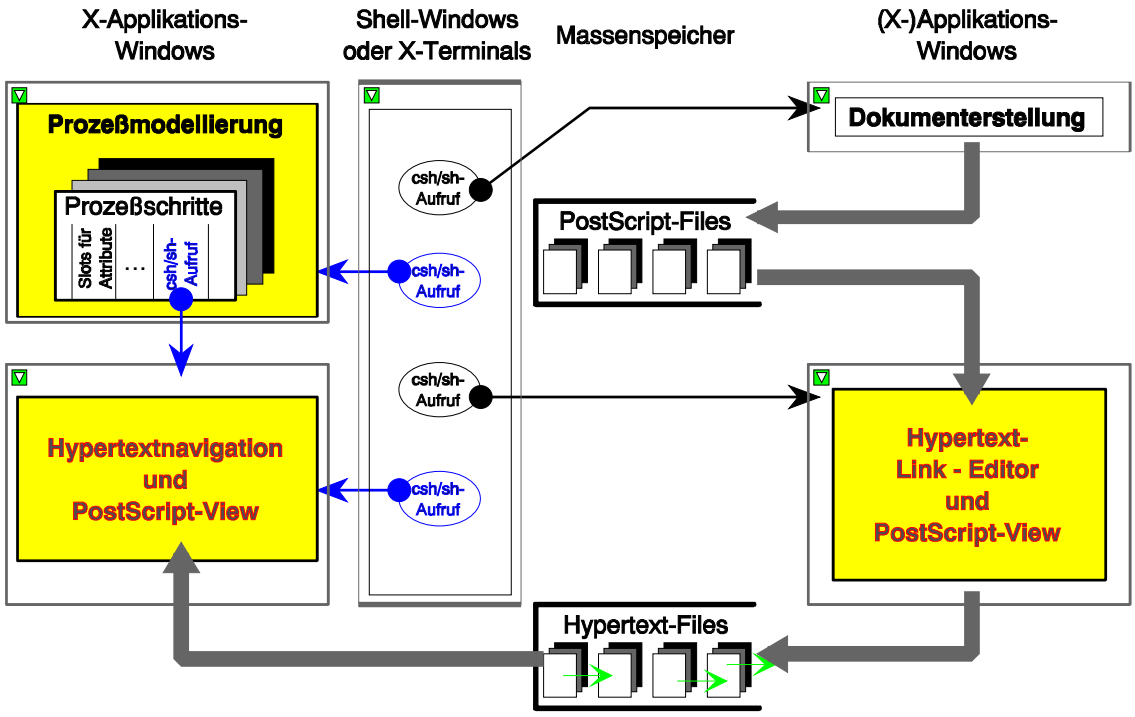


*Abb. 2: Die Komponenten des Hypertextsystems und ihr Zusammenwirken*

Abbildung 2 zeigt die konzipierten Komponenten und ihr Zusammenwirken schematisch.

## 1.4 Motivation: Hypertext in der LIGA–Konstruktion

Es stellt sich die Frage, in wieweit ein Hypertextsystem in LIGA–Konstruktion und LIGA–Fertigung sinnvoll einzubringen ist. Zur Klärung dieser Frage muß das für die Hypertextvernetzung in Frage kommende Schrifttum betrachtet werden. Aus Sicht des Konstrukteurs kann es in fünf Klassen geteilt werden:

(0) Anleitungen und Beschreibungen zu dem für die Konstruktion eingesetzten Softwaresystem – sogenannte „On–Line–Hilfe" für die eingesetzten ***Konstruktionssysteme.***

(1) Beschreibungen und schriftlich niedergelegte *Heuristiken* für bestimmte ***Konstruktionsaufgaben,*** die eigene oder fremde Erfahrungen der Konstrukteure und Alternativen in bestimmten Entwurfsphasen wiedergeben.

(2) Beschreibungen und Anleitungen zu ausgewählten ***Teilprozessen*** und speziellen ***Prozeßschritten,*** die sich nicht als Attribute im konventionellen Sinn dem Prozeßschritt oder Teilprozeß zuordnen lassen oder deren Zuordnung als Attribute des Prozeßschrittes oder Teilprozesses aus Expertensicht nicht sinnvoll ist.

(3) Beschreibungen und Handbücher zu den – ggfs. als Ausrüstungsgegenstände modellierten – ***Fertigungswerkzeugen,*** die in einem speziellen Prozeßschritt oder Teilprozeß nicht bei der Konstruktion, sondern bei der Fertigung zum Einsatz kommen.

(4) ***Publikationen zusammenfassender Art,*** die weder einer Entwurfsaufgabe noch einem Fertigungsprozeß direkt zugeordnet werden können.

Eine Vernetzung dieses Schrifttums in einem Hypertextsystem bietet verschiedene Vorteile:

Zunächst ist allein der datenhalterische Aspekt zu verzeichnen: durch eine konsequent und überlegt gestaltete Hypertextvernetzung werden entsprechende Dokumentationen, insbesondere aus der Klasse (4), systematisch erfaßbar. Ihre inhaltliche Verwandtschaft zu anderen Schriften wird durch die Verbindung mit Hypertextlinks transparent, und das Auffinden verwandter Dokumente und interessanter Abschnitte ist rechnergestützt möglich und nicht mehr vom Erinnerungsvermögen einzelner Personen abhängig.

Darüber hinaus bietet ein Hypertext–Netz dem Konstrukteur die Möglichkeit, im Entwurfsprozeß direkt am Entwurfswerkzeug „Bildschirm" Dokumente (Klassen (1) und (2)) zu bestimmten Schlagworten rasch zu finden und zu sichten. Über die Technik der Hypertextverbindungen können verwandte Probleme und eventuell alternative Lösungen für den Entwurf gefunden werden.

Der Erfahrungsaustausch zwischen den Konstrukteuren wird gefördert, wenn entsprechende Notizen und Dokumente aller an der Konstruktion beteiligten Personen rasch in das Netz eingebunden werden.

Der Fertigungstechniker, der einen Prozeß am Bildschirm in der Modellierung verfolgt, hat über das Hypertextsystem die Möglichkeit, rasch auf ggfs. zu einem bestimmten Teilprozeß vorhandenen Erfahrungen (Klassen (2) und (3)) von sich selbst und Kollegen zurückzugreifen, ohne erst das entsprechende, gedruckte Material beschaffen und durchsehen zu müssen.

Die Einarbeitung neuer Mitarbeiter kann über ein Hypertextsystem effektiv durchgeführt werden, da die Auswahl der Literatur nicht mehr allein vom Geschick des Mitarbeiters beim Auffinden der relevanten Literatur abhängt, sondern durch die

vorbereitende Arbeit des erfahrenen Hypertextredakteurs im Rechner fixiert gestützt wird.

Die voranstehende Liste der Vorteile ist nicht vollständig, verschiedene weitere Gründe, die für den Einsatz eines Hypertextsystems sprechen, ließen sich anfügen.

Was bei dieser Betrachtung jedoch deutlich wird, ist, daß bei der Erstellung des Hypertextnetzes durch die Hypertextredakteure große Sorgfalt und fachliche Kompetenz erforderlich ist. Die richtige Verbindung der Dokumente und Textabschnitte entscheidet über die Akzeptanz des gesamten Systems. Wird die redaktionelle Arbeit nicht sorgfältig durchgeführt, beschränkt sie sich beispielsweise auf die reine Vernetzung aufgrund von konventionellen Referenzen (Inhalts–, Abbildungs– und Schlagwortverzeichnisse, ausdrückliche Querverweise) bietet sie gegenüber dem gedruckten Medium keine signifikanten Vorteile.

Es wird ebenso deutlich, daß die Verfügbarkeit des Dokument–Materials in einem portablen Datenformat von entscheidender Bedeutung ist. Bei dem großen Umfang des zu vernetzenden Schrifttums müssen aufwendige Konvertierungen und Neuerfassungen zu den Ausnahmefällen gehören. Ebenso muß aus Gründen des Umfangs der Daten vermieden werden, für unterschiedliche Ausgabegeräte unterschiedliche Versionen der Dokumente zu halten. Die zentrale Entscheidung für ein einheitliches, weitverbreitetes, quell– und ausgabeunabhängiges Dokumentformat in dieser Studienarbeit wird dadurch verständlich.

*Die vorliegende Arbeit wird in den nächsten beiden Abschnitten zugrundeliegende Voraussetzungen klären, zunächst im Abschnitt 2 mit einer knappen Betrachtung des Prinzips „Hypertext" und dann in Abschnitt 3 mit einem groben Überblick über das als einheitliches Dokumentformat bestimmte „PostScript".*

*In den anschließenden Abschnitten 4 und 5 wird dann auf die eigentliche Realisierung eingegangen, zunächst auf die Navigationskomponente (Abschnitt 4) und dann auf das Werkzeug des Hypertextredakteurs, den „Link–Editor" (Abschnitt 5). Hier steht die Beschreibung der Umprogrammierung im Zentrum, die jedoch nicht auf programmiersprachliche Einzelheiten eingehen, sondern das algorithmische Konzept vorstellen soll.*

*Den Abschluß bilden Überlegungen zur Arbeit des Hypertextredakteurs und zum Einsatz des Hypertextnetzes in Abschnitt 6.*

# 2 Hypertext – eine einführende Betrachtung

## 2.1 Informationsvermittlung mit herkömmlichen Textmedien

Die Vermittlung sachbezogener Informationen war bislang die Domäne der klassischen Printmedien, allen voran Bücher und Veröffentlichungen in Fachzeitschriften.

Die für diese Betrachtung kennzeichnenden Eigenschaften der Mehrzahl dieser Printmedien sind:

- ❏ Die dargestellte Information ist statisch und einer raschen Änderung oder Anpassung an veränderte Sachverhalte nicht oder nur schwer zugänglich.
- ❏ Die Information wird in *einer*, vom Verfasser festgelegten, linearen Reihenfolge präsentiert. Diese Reihenfolge kann vom Leser zwar verlassen werden (überspringen, zurückblättern, ...), ihm stehen dabei im allgemeinen jedoch keine oder nur schwach ausgeprägte Mechanismen zum Auffinden geeigneter Sprungziele zur Verfügung. Der Autor hat keine Möglichkeit, die Sprünge des Lesers explizit zu beeinflussen.

Die Nachteile des ersten Merkmals der Printmedien können dadurch behoben werden, daß man statt des Datenträgers Papier entsprechende Softwarewerkzeuge und Datenverarbeitungsanlagen einsetzt, um die Information zu präsentieren.

Die Nachteile des zweiten Merkmals können durch geeignete Mechanismen behoben werden, die einem Autor schon bei der Konzeption seiner Informationspräsentation ermöglichen, verschiedene Pfade durch das Material vorzusehen.

Erste, rudimentäre Ansätze zur Aufweichung der strikt linearen Informationsdarreichung und zur Einflußnahme auf die Aufnahmepfade des Lesers durch den Autor finden sich seit Alters her in Büchern: Inhalts- und Schlagwortverzeichnisse oder Verweise auf verwandte Artikel in Lexika sind einfachste Techniken zur individuell angepaßten Aufnahme von ausgewählten Informationen aus den gedruckten Medien. Daneben existieren überwiegend in der Sekundärliteratur, z. B. in Bibliographien, Verzeichnissen, Führern und Wörterbüchern, Medien, die nur zum Zwecke der gezielten Verzweigung und nicht zum linearen Lesen konzipiert sind.

Ansätze, die Verzweigungstechnik auch in die „lineare" Primärliteratur dominierend einzubringen, wurden in den Siebziger Jahren vor allem im Schulunterricht mit den sogenannten „Lernprogrammen" gemacht, die – zwar noch auf Druckmedien – nach der Darreichung gewisser Informationspakete durch Testfragen die Aufnahme des vermittelten Stoffes überprüften und bei der unmittelbar anschließenden Auswertung der Tests durch den Leser selbst entsprechende Rücksprünge oder Vorwärtssprünge im Unterrichtsmaterial veranlaßten.

## 2.2 Hypertext

Der wesentliche Aspekt eines Hypertextsystems ist das *geführte* Verlassen der durch den Text vorgegebenen linearen Präsentationsreihenfolge der Information.

Bereits in den Überlegungen des vorangehenden Abschnitts wird als wichtiger Mechanismus der Sprung innerhalb eines Textes dargestellt. Eine Hypertextanwendung erhebt durch entsprechende Mechanismen das Verfolgen dieser Sprünge auf einem individuellen, an den Bedürfnissen des Lesers orientierten Pfad zum zentralen Charakteristikum.

Der Mechanismus selbst ist dadurch realisiert, daß vorgesehene *Startstellen* im Text durch geeignete Auszeichnung hervorgehoben werden. Der Benutzer kann durch bestimmte Interaktionen (Mausklick, Tastencode, ...) eine solche Startposition aktivieren und den Sprung zu dem für den Startpunkt festgelegten *Ziel* auslösen. Da auf diese Weise jeweils eine Startposition im laufenden Text mit einer Zielposition *verbunden* wird, nennt man den charakteristischen Hypertextmechanismus auch einen *Link*[2].

Erweiternd zu den ersten, bereits genannten Ansätzen tritt in Hypertextanwendungen die Vernetzung verschiedener Schriftstücke hinzu: Dazu ist der Link–Mechanismus dahingehend erweitert, daß nicht nur Sprungziele im selben Hypertextdokument angesteuert werden können, sondern daß auch verfügbare andere Dokumente als Ziel eines Links angezeigt werden.

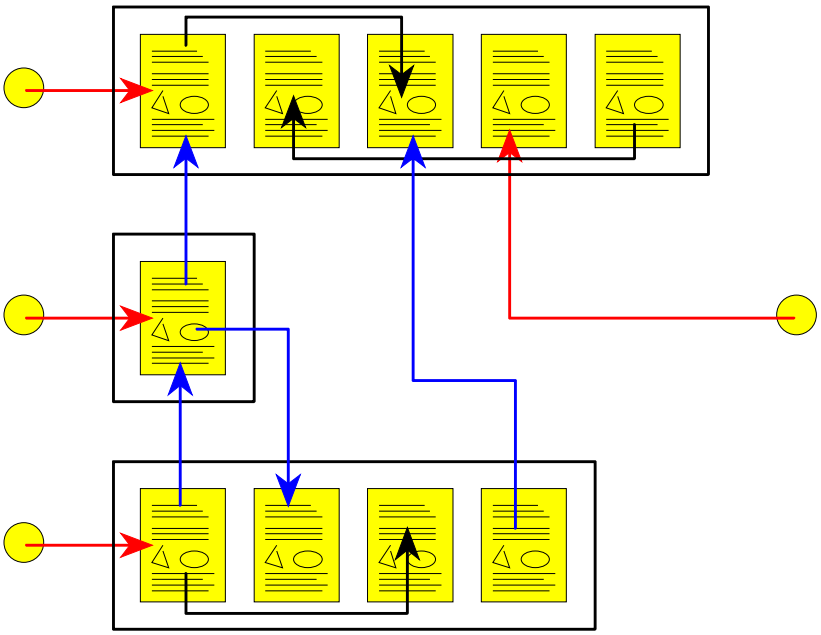

*Abb. 3: Schema eines typischen Hypertext–Netzes*

Daraus ergibt sich für einen *Hypertextautor* eine höhere Anforderung als für den konventionellen Textautor: Neben der Festlegung der konventionellen Textpräsentation

[2] Link, von englisch: link — Kettenglied, to link — verknüpfen, verketten

müssen Links in den Texten eingebaut werden, es sind Links zwischen verschiedenen Texten vorzusehen und es sind geeignete Einsprungstellen in das Netz zu schaffen. Ein Link, der innerhalb *eines* Dokumentes springt, wird im weiteren Text als ***Intralink*** bezeichnet, eine Hypertextverbindung, die aus einem Dokument hinaus in ein weiteres führt, analog als ***Interlink***.

Ein schematisches Szenario dieser Art zeigt Abbildung 3. Hierbei sind durch die Pfeile Intra– (schwarz) und Interlinks (heller) gekennzeichnet, die Kreise bezeichnen die vorgesehenen Eintrittsstellen in das Netz, und die „natürliche" Verbindung der einzelnen Seiten (seitenweise Vorwärts– und Rückwärtsblättern) in einem Dokument wird durch die Umrandung zusammengehöriger Seiten dargestellt.

## 2.3 Hypermedia

Hypertextanwendungen sind darauf beschränkt, Darstellungen zu verarbeiten, die in klassischer Weise gedruckt werden können. Das Morphem „–text" darf dabei nicht zu eng verstanden werden, denn selbstverständlich können auch Grafiken als Startpunkt und Ziel eines Links fungieren. Hypertext hat beispielsweise bei der Betrachtung von Konstruktionsplänen mit Gesamtansichten und Explosionszeichnungen[3] verschiedener Details ein ideales Einsatzgebiet.

Die Ansätze des Hypertextkonzepts regen aber dazu an, die Technik auf andere Medien als Text und Grafik zu erweitern; das Konzept wird dann entsprechend als *Hypermedia* bezeichnet.

In einer Hypermediaanwendung kann es etwa sinnvoll sein, als Ziel eines Links nicht eine bestimmte Textstelle aufzuschlagen, sondern statt dessen beispielsweise ein erläuterndes Video ablaufen zu lassen oder eine Sprachanmerkung akustisch auszuführen. Auch ist denkbar, eine ausgezeichnete Grafik auf dem Bildschirm zu animieren oder anderes mehr.

## 2.4 Graphentheoretische Einordnung von Hypertext

Als namengebendes Kennzeichen bleibt jedem *„Hyper..."*–System die klassifizierende Ausbildung von Präsentationspfaden in einer bestehenden Menge von Informationen.

Es ist daher möglich, die Definition des *Hypergraphen* aus der Graphentheorie zur Definition eines Hypermediasystems zu adaptieren. Diese liest man etwa in /Schneider91/ wie folgt: „Ein Graph besteht aus *Knoten* und Verbindungen zwischen Ihnen, den *Kanten*. Abstrakt lassen sich die Kanten als Paare von Knoten (Anfangs– und Endknoten) auffassen. Ein Hypergraph besteht aus Knoten und einer Verallgemeinerung der Kanten,

[3] Eine Explosionszeichnung eines Gegenstandes zerlegt diesen in seine einzelnen Baugruppen oder Bauteile, die in einer in etwa den montierten Positionen entsprechenden Anordnung unverbunden dargestellt werden.

den Hyperkanten. Eine Hyperkante ist eine beliebige Menge von Knoten. Beispiel: Die Haltestellen in einem Straßenbahnnetz einer Großstadt können als Knoten eines Hypergraphen aufgefaßt werden. Die Hyperkanten sind die Haltestellen einer Linie. Alle Knoten, die in einer Hyperkante liegen, sind in der Realität durch eine Linie, ohne Umsteigen verbunden. Zwei Hyperkanten inzidieren, wenn sie gemeinsame Knoten besitzen. In der Realität bedeutet dies, daß zwei Linien gemeinsame Haltestellen zum Umsteigen aufweisen."

Ein Hypertext– oder Hypermediadokument kann als Graph oder sogar *Digraph*[4] aufgefaßt werden: Als Knoten des Hypermedia sind je nach Granularität des eingesetzten Systems Wörter, Grafiken, Sätze, Absätze oder Seiten zu sehen. Die Aneinanderreihung dieser Knoten gemäß der Ordnung im Druckbild liefert die elementaren Kanten. Die Gliederung in verschiedene Dokumente partitioniert den gesamten Textgraphen in Untergraphen, die durch *spannende Wege*[5] aus elementaren Kanten aufgespannt werden. Die Hyperkanten schließlich sind alle Wege, die sich durch Kombination von Elementarkanten und Links ergeben. In der besonderen Ausprägung als Interlinks bilden sie im Gesamtgraphen die *Brücken*[6], die Ziele der Interlinks stellen die *Artikulationspunkte*[7] des Gesamtgraphen dar.

Die Betrachtung eines Hypermedianetzes als Graph, die nicht weiter verfolgt werden soll, gibt für die Analyse und die Beherrschung der Navigation in diesen Dokumenten und die Handhabung der Hypertext–Datenstrukturen das reiche Instrumentarium der Algorithmen aus der Graphentheorie an die Hand.

---

[4] Digraph, von englisch: directed graph — gerichteter Graph, bei dem die Richtung der Kanten vom Ursprungsknoten zum Zielknoten wesentlich ist.

[5] Ein spannender Weg enthält alle Knoten eines Graphen

[6] Eine Kante, durch deren Wegnahme ein Graph unzusammenhängend wird, heißt Brücke

[7] Ein Knoten, durch dessen Wegnahme ein Graph unzusammenhängend wird, heißt Artikulationspunkt des Graphen. Ein Graph heißt zusammenhängend, wenn von jedem Knoten k aus jeder andere Knoten $k' \neq k$ erreichbar ist.

# 3 Grundlagen der Seitenbeschreibungssprache PostScript

Wegen der elementaren Bedeutung von *PostScript* bei der Durchführung dieser Studienarbeit sei das Konzept dieser Programmiersprache nachfolgend knapp beschrieben.

## 3.1 Das Konzept von PostScript

PostScript ist eine seit 1976 entwickelte und nach der Gründung von Adobe Systems Incorporated in 1982 erstmals unter dem jetzigen Namen vertriebene Programmiersprache zum besonderen Zweck, den grafischen Aufbau einer Seite unabhängig vom eingesetzten Ausgabegerät zu beschreiben. Sie ist in den letzten Jahren zum De–facto–Standard für das professionelle Generieren von Druckausgaben geworden.

PostScript ist eine interpretative Sprache zur Beschreibung zweidimensionaler Grafiken. Das gewünschte Ausgabegerät wird direkt mit PostScript programmiert und muß den Interpreter für die Sprachkonstrukte in der Hardware bereitstellen. Aufgabe des Interpreters ist, die abstrakte Beschreibung der Ausgabegrafik umzusetzen in Punktraster oder Stiftbewegungen des jeweiligen Ausgabegerätes. Dabei werden erst bei der Umrechnung die Auflösung des Ausgabegerätes und seine Farbfähigkeiten berücksichtigt, so daß eine PostScript–Ausgabe immer die bestmögliche Darstellung der Grafik auf dem gewählten Ausgabemedium ergibt. Anpassungen an Hardwareeigenheiten müssen sämtlich im Interpreter residieren, sie finden keinen Eingang in das ausgeführte Programm. Da die Übersetzung als Interpreter in Hardware realisiert ist, entstehen keine „Laufzeit–PostScript–Programme", die dann wieder geräteabhängig wären.

Ist ein Hardware–Interpreter nicht verfügbar (etwa bei der Ausgabe auf Bildschirmen), kann ein Software–Interpreter die übersetzende Funktion ausführen. In diesem Fall können je nach Produkt auch geräteabhängige Druckdateien entstehen, was zwar der Philosophie von PostScript widerspricht, aber für die praktische Anwendung sehr von Bedeutung ist. Der bekannteste Vertreter der PostScript–Software–Interpreter ist das Produkt *„GhostScript"*, eine Entwicklung der Aladdin Enterprises aus 1990, die von der Free Software Foundation als Freeware unter der GNU–Lizenz[8] erhältlich ist. GhostScript dient bei der vorliegenden Arbeit als Basis für den Link–Editor.

[8] Die "GNU General Public License" der Free Software Foundation gewährt den Einsatz und die Redistribution der davon betroffenen Programme. Sie läßt außerdem eine Veränderung der Programme unter Einhaltung gewisser Regeln zu. Sie ist im englischen Originaltext im Anhang dieser Arbeit unter Abschnitt 7.2 abgedruckt.

Ein PostScript–Programm — selbst für die komplexeste Grafik — kann unter dieser Voraussetzung als reiner ASCII–Text abgespeichert werden und ist damit prädestiniert für die Portierung auf verschiedene Systeme.

Die hervorzuhebenden Merkmale von PostScript sind:

- Die Generierung von beliebigen *„freien Formen"*, die aus Strecken, Bögen und kubischen Kurven gebildet werden können. Die Formen können sich überschneiden, aus unverbundenen Bereichen gebildet sein und Lücken aufweisen.
- *Painting–Primitive* ermöglichen, eine beliebige Form mit einer Umrandungslinie beliebiger Dicke zu versehen, sie mit Farbe zu füllen oder als begrenzendes, „ausstanzendes" Fenster auf eine beliebige Grafik zu setzen.
- Der *Textsatz* ist die am weitesten verbreitete Domäne von PostScript. Text läßt sich vollständig in die programmierte Grafik integrieren, da alle Zeichen eines Fonts[9] als *freie Form* behandelt werden und somit jedem PostScript–Operator zugänglich sind. Die Datenformate, die hierbei zur Definition von Fonts Verwendung finden, haben sich weltweit (neben „TrueType") durchgesetzt.
- Die *Integration von Bitmap–Grafiken,* etwa aus Fotografien oder durch andere Software generiert, ist möglich. Mehrere Mechanismen zur Gestaltung dieser Bitmap–Grafiken sind eingebunden.
- Es liegt ein *generelles Koordinatensystem* für die Programmierung zugrunde, auf dem verschiedene Einrichtungen zur Durchführung von linearen Transformationen wie Verschiebung, Spiegelung, Streckung verfügbar sind. Die Transformationen können auf allen grafischen Elementen durchgeführt werden.

Einen Eindruck dieser mächtigen Fähigkeiten sollen wenige, ausgewählte Beispiele im nächsten Abschnitt vermitteln.

## 3.2 Ausgewählte PostScript–Konstrukte und Beispiele

### 3.2.1 Der PostScript–Stack und einfache Operatoren

Obwohl PostScript als Seitenbeschreibungssprache konzipiert ist, kann sie in Grenzen auch als „normale" Programmiersprache eingesetzt werden. Ein Syntaxprinzip läßt sich

[9] Als Font, zu deutsch Schriftschnitt, bezeichnet man eine Sammlung von verschiedenen Schriftzeichen eines einheitlichen Erscheinungsbildes. Beispielsweise ist der Fließtext in dieser Arbeit im Font „Garamond" — in verschiedenen Schriftgrößen und z. T. mit Attributen (kursiv, fett, ...) versehen — gesetzt, die Programmausschnitte in dem „Schreibmaschinen"–Font „Courier" und die Beschriftungen in allen Abbildungen, die nicht als Schnappschuß vom Bildschirm genommen wurden, im Font „Helvetica".

dabei bereits aus dem Namen ablesen: PostScript arbeitet mit einem *Operandenstapel* (LIFO[10]–Datenstruktur) nach der *Postfixnotation*.

Daten werden in diesen *Stack* durch Hintereinanderschreiben in einer zu interpretierenden Zeile des PostScript–Programmes eingereiht („push"). Der nächste, vom Interpreter erkannte Operator nimmt sich die von ihm benötigten Daten gemäß dem LIFO–Prinzip vom Stack.

Sollen beispielsweise die beiden Konstanten 3 und 5 addiert werden, geschieht dies in PostScript durch die Zeile

```
3    5    add
```

Die Bildung von Ausdrücken geschieht entsprechend, sie ist jedem Benutzer eines Taschenrechners mit UPN (Umgekehrter Polnischer Notation) vertraut:

```
25   4    sub  3    div  3    4    add  mul
9    4    10   mul  add
```

liefern als Ergebnis jeweils 49, im ersten Fall als Auswertung des Terms $\frac{(25-4)}{3} \cdot (3+4)$, im zweiten Fall als Ergebnis des Terms $9+4 \cdot 10$.

Bei der Evaluation wird die Programmzeile von links nach rechts gelesen, wobei Operanden auf den Stapel gelegt werden. Bei Erreichen eines Operators werden jeweils die benötigten Operanden vom Stack genommen und das Ergebnis der gerade ausgeführten Operation wieder (oben) auf dem Stack abgelegt.

Neben den arithmetischen Operatoren `add`, `sub`, `div`, `idiv`, `mod`, `mul` und `neg` stehen auch spezielle Operatoren zur Handhabung des Stacks zur Verfügung: `exch` vertauscht die beiden obersten Elemente auf dem Stack, `clear` löscht den gesamten Stack, `dup` dupliziert das oberste Element auf dem Stack und `pop` löscht das oberste Element. Zwei Operatoren liefern Ausgabe auf dem Standard–Ausgabegerät („*stdout*" in UNIX, was nicht das PostScript–Ausgabemedium sein muß): „==" schreibt das oberste Element aus dem Stack und führt darauf ein `pop` aus, `pstack` schreibt den Inhalt des gesamten Stacks aus, ohne daran Änderungen durchzuführen. Bei der Ausgabe werden Objekte der einfachen Datentypen wie `zahl`, `string` und `array` als Werte angezeigt, alle nicht einfach darzustellenden Objekte werden durch einen Typbezeichner repräsentiert, wie z. B.
„–`dictionary`–" oder „–`file`–".

### 3.2.2 Liniengrafik, Painting und Textsatz

Liniengrafik wird in PostScript ähnlich wie bei der „*Turtlegraphics*" in intuitiven Programmiersprachen wie „Logo" programmiert: ein imaginärer Stift zieht seinen Pfad auf der virtuellen Seite. Durch entsprechende Operatoren wird entweder seine Zielposition absolut festgelegt oder relativ zu einer Ausgangsposition angegeben. Durch die Wahl entsprechender Ausprägungen der grundlegenden zwei Positionierungsoperatoren wird die Sichtbarkeit einer Spur bestimmt.

---

[10] LIFO, von englisch "Last In – First Out": Datenstruktur, die das zuletzt eingestellte Element als erstes wieder ausgibt.

Eine Gerade wird z. B. durch das folgende kleine Programm[11] auf dem Ausgabemedium generiert:

```
newpath
     144  72   moveto
     144  432  lineto
stroke
showpage
```

Der Operator `newpath` eröffnet dabei einen neuen Zeichnungspfad. Der imaginäre Stift wird dann zunächst mit `moveto` zu einer fixen Koordinate (x– und y–Koordinate vom Stack) bewegt, ohne eine sichtbare Spur zu hinterlassen.

Im Standard–Koordinatensystem von PostScript hat die linke, untere Ecke der virtuellen Seite die Koordinaten (0, 0), die x–Achse läuft nach rechts und die y–Achse nach oben. Die Einheit des Koordinatensystems ist $\frac{1}{72}$" oder ein „Punkt"[12] in x– und y–Richtung.

Der `lineto`–Operator in der dritten Anweisung bewegt den imaginären Stift geradlinig von der augenblicklichen Position zu der in den beiden obersten Werten auf dem Stack angegebenen Koordinate (144, 432). Erst durch den Operator `stroke` der vierten Programmzeile wird der zuvor gebildete Pfad auch gezeichnet, bleibt jedoch auf dem Ausgabemedium noch solange unsichtbar, bis die komplette, eventuell aus vielen Pfaden komponierte Seite durch den `showpage`–Befehl gedruckt, beziehungsweise angezeigt wird.

Die bereits angesprochenen, relativen Varianten der Befehle `moveto` und `lineto` heißen `rmoveto` und `rlineto`. Die bei ihrer Ausführung vom Stack genommenen Operanden werden auf die Koordinaten der augenblicklichen Stiftposition addiert.

Die Gestalt des generierten Pfades läßt sich mit den Operatoren `setlinewidth`, `setgray`, `fill` und `stroke` beeinflussen: `setlinewidth` nimmt einen Wert vom Stack, der die Strichdicke im Punktmaß angibt, `setgray` nimmt einen Wert $0 \leq w \leq 1$ vom Stack, der die Schwärzung in Prozent angibt. `stroke` zieht den Pfad in der gewählten Strichdicke aus, `fill` füllt die durch den Pfad und seine eventuelle geradlinige Schließung begrenzte Fläche im gewählten Grauwertmuster aus.

Die Operatoren des Textsatzes in PostScript wirken zum überwiegenden Teil auf Zeichenketten, *Strings*. Ein String ist eine Aneinanderreihung von druckbaren Zeichen und wird in PostScript durch runde Klammern begrenzt.

Ein wesentlicher Aspekt des Schriftsatzes ist die Wahl des Fonts, in dem der betroffene Text ausgegeben werden soll. Der Operator zur Wahl eines Fonts ist `findfont`. Die Größe, in der Text gesetzt wird, wird mit `scalefont` festgelegt, Einheit ist Punkt. Der ausgewählte Font wird in der festgelegten Punktgröße mit dem Operator `setfont` aktiviert. Ein gesetzter Text wird mit `show` auf der laufenden Seite ausgegeben, und mit

---

[11] Die hier abgedruckten Programme können bei Verfügbarkeit eines interaktiven PostScript–Interpreters mit passendem Bildschirmtreiber (wie z. B. GhostScript) direkt eingetippt und ihre Effekte auf dem Monitor betrachtet werden.

[12] Das Zeichen „" steht für das englische Längenmaß „inch"; $1\ inch \approx 2{,}54\ cm$. In der Drucktechnik ist als Größeneinheit das Maß „Punkt" verbreitet. Ein drucktechnischer „Punkt" hat die Ausdehnung $\frac{1}{72}$"; die gängige Abkürzung dieser Maßeinheit ist „pt".

`stringwidth` kann die Ausdehnung eines gesetzten Strings in Koordinateneinheiten ermittelt werden.

```
/Times-Roman findfont
18 scalefont
setfont
72 144 moveto
(Hypertext) show
showpage
```

schreibt das Wort „Hypertext" 1" vom linken Rand und 2" vom unteren Rand der Seite beginnend im Font „Times–Roman" in 18 pt Größe auf das Ausgabemedium.

### 3.2.3 Variable und Prozeduren

Um Flexibilität zu erreichen müssen Programmiersprachen ermöglichen, veränderliche Werte in fixierte Abläufe einzubeziehen. Dies geschieht in den meisten Programmiersprachen durch ein Variablenkonzept, das symbolische Bezeichner für bestimmte Speicherbereiche vorsieht, in denen zur Programmausführung entsprechende Werte eingestellt und ggfs. während der Programmausführung verändert werden.

Die Zuordnung von Bezeichnern zu Werten geschieht in PostScript in sogenannten *Dictionaries,* die Bezeichner in den Dictionaries heißen *Keys* und die zugeordneten Werte entsprechend *Values*. In PostScript werden automatisch wenigstens zwei Dictionaries angelegt, ein *System Dictionary*, das für jeden Operator entsprechende Aktionen definiert, und das *User Dictionary,* in dem benutzerdefinierte Variablen gehalten werden.

Der Interpreter sucht Symbole zunächst im User Dictionary, danach im System Dictionary, so daß der Benutzer Systemvorgaben durch eigene Definitionen überdecken kann. Es ist möglich, noch weitere Dictionaries in Programmen anzulegen; sie werden insgesamt in einem eigenen Stack gehalten und dort vom Interpreter jeweils „top–down" nach Keys durchsucht.

Zur Einbringung von Paaren in ein Dictionary steht der Operator `def` zur Verfügung.

Soll beispielsweise einer Variable mit dem Bezeichner PI der Wert 3,1415926 zugeordnet werden, geschieht das mittels der Programmzeile

```
/PI  3.1415926 def
```

Der Slash vor dem Bezeichner dient dabei als Fluchtsymbol, das die Suche des Interpreters nach einem bereits vorhandenen Bezeichner unterdrückt.

Die Handhabung von Prozeduren geschieht in PostScript analog zur Handhabung von Variablen: Prozeduren werden in einem gesonderten Dictionary gehalten und stellen unter einem Bezeichner ein Folge von Operationen oder weiteren Prozeduraufrufen zur Verfügung. Die Prozedur wird dann im Programm als quasi „neuer", selbstdefinierter Operator eingesetzt.

Beispielsweise muß später in der Navigationskomponente an vielen Stellen ein Rechteck aus vier in einer festgelegten Reihenfolge auftretenden x– und y–Koordinatenwerten gezeichnet werden. Dazu ist die Definition einer Prozedur LRE (für „Link–Rechteck") sinnvoll, die folgendermaßen aussehen könnte[13]:

```
/LRE {
% ************************************************
% zeichnet ein Rechteck aus x1, y1, x2, y2 vom Stack:
% (x1, y2) +--------------------------------+ (x2, y2)
%          |                                 |
%          |                                 |
%          |                                 |
% (x1, y1) +--------------------------------+ (x2, y1)
% ************************************************

             newpath                 % legt neuen Pfad an
                                     % Stack: x1 y1 x2 y2
                  exch 4 2 roll      % y2 x2 x1 y1
                  exch 4 -2 roll     % y1 x1 y2 x2
                  dup  3 2 roll      % y1 x1 x2 x2 y2 y2
                  dup  4 1 roll      % y1 x1 y2 x2 x2 y2
             moveto                  % r.o. Ecke als Start
                                     % Stack: y1 x1 y2 x2
                  dup  5 -1 roll     % x1 y2 x2 x2 y1
                  dup  6 1 roll      % y1 x1 y2 x2 x2 y1
             lineto                  % r.u. Ecke ansteuern
                                     % Stack: y1 x1 y2 x2
                  4 -2 roll dup      % y2 x2 y1 x1 x1
                  5 1 roll exch      % x1 y2 x2 x1 y1
             lineto                  % l.u. Ecke ansteuern
                                     % Stack: x1 y2 x2
                  exch dup           % x1 x2 y2 y2
                  4 -1 roll exch     % x2 y2 x1 y2
             lineto                  % l.o. Ecke ansteuern
                                     % Stack: x2 y2
             lineto                  % zurück zu r.o.
                                     % Stack: (empty)
             closepath               % r.o. Ecke schließen
             stroke                  % Linie zeichnen
        } def                        % Prozedur definieren
```

Jeder Text, der in PostScript nach dem Zeichen „%" eingegeben wird, zählt bis zum Zeilenende als Kommentar. Der verwendete Operator `roll` permutiert Elemente auf dem Stack und benötigt dazu zwei Operanden. Er entnimmt dem obersten Operanden die Bewegungsrichtung und Schrittweite (d < 0, um d Schritte nach links; d > 0, um d Schritte nach rechts), dem nächsten die Anzahl der Elemente vom neuen Top an gerechnet, die permutiert werden sollen.

Die oben definierte Prozedur kann jetzt an jeder Stelle im PostScript–Code unter ihrem Namen LRE aufgerufen werden; sie benötigt dann auf dem Stack vier numerische Werte, die von links nach rechts als x–Ko-

[13] Das abgedruckte Programm ist in keiner Weise optimal, jedoch korrekt. Es extrahiert die Koordinaten der anzusteuernden Ecken in der Reihenfolge von oben, rechts im Uhrzeigersinn aus dem Stack und fährt die Ecken danach an.

ordinate der unteren Ecken, y–Koordinate der linken Ecken, x–Koordinate der oberen Ecken und y–Koordinate der rechten Ecken des zu zeichnenden Rechtecks Eingang finden:

```
% weiter unten im PostScript-Programm
92 264 229 284 LRE
```

### 3.2.4 Ablaufsteuerung

Auch in PostScript bestehen die üblichen Möglichkeiten, den rein sequentiellen Programmablauf zu verlassen.

Die Gewinnung boolescher Ergebnisse für die Verzweigungs– oder Schleifensteuerung geschieht mittels der Vergleiche `eq` (=), `ne` (≠), `gt` (>), `lt` (<), `ge` (≥) und `le` (≤). Boolesche Ausdrücke lassen sich mit den bekannten Junktoren `not` (¬), `and` (∧), `or` (∨) und `xor` (≢) bilden.

Zur bedingten Ausführung einer Prozedur oder Sequenz steht der Operator `if` zur Verfügung, die Alternativenbildung geschieht mit `ifelse`. An Schleifenstrukturen existiert der Operator `repeat`, der die Operation an der Top–Position des Stacks so oft ausführt, wie es die Zahl an zweiter Stelle im Stack definiert, aber auch die Zählschleife `for` gehört zum Sprachumfang von PostScript.

## 3.3 Die Dokumentstrukturierungsregeln von Adobe

Üblicherweise wird ein PostScript–Programm weder von einem Programmierer geschrieben noch von Menschen gelesen. Die Erstellung geschieht mittels Druckertreibern oder Umsetzungsprogrammen aus den zugrundeliegenden Textverarbeitungs– oder Grafikanwendungen; der PostScript–Quelltext wird in der Mehrzahl direkt oder indirekt in eine Satzmaschine oder einen Drucker eingespeist. In diesem Fall ist es sinnvoll, den Code zur Einsparung von Übertragungszeit und Speicher kompakt zu halten und weder durch eine übersichtliche, strukturierte Zeilengliederung noch durch erläuternde Kommentare zu gestalten.

Um aber dennoch ein maschinell generiertes PostScript–Programm für Debugging–Zwecke lesbar zu machen, oder auch, um für manche Druckerspooler ein vorbereitendes Bereitstellen von Ressourcen (Fonts, Seitenpuffer, ...) zu ermöglichen, hat die „Adobe Systems Incorporated" *Strukturierungskonventionen* entwickelt und in /Adobe90a/ publiziert.

Code, der den Strukturkonventionen genügt wird als *konform,* englisch „conforming", bezeichnet, anderer Code entsprechend als *nonkonform,* englisch „non conforming". Die Nonkonformität schließt jedoch nicht unbedingt die Ausführung des Programms durch einen Interpreter aus. In vielen Druckertreibern kann deshalb als Option die Generierung von PostScript–Code nach der „Adobe–Document–Structure" gewählt werden. Die Konformität selbst kann in verschiedener Güte vorliegen. In der nachfolgenden Beschreibung sind alle Strukturierungskommentare, die für eine minimale Konformität notwendig sind, durch das Zeichen „☑" gekennzeichnet.

Die grundlegende Struktur–Empfehlung fordert eine Zweigliederung des Codes in ein *„Prologue" (Vorspann)* und ein *„Script"*, was als Analogon zu Wirths[14] „Deklarationsteil" und „Aktionsteil" in PASCAL zu sehen ist: im Prologue werden alle globalen Eigenschaften, Prozeduren und Variablen für das Programm festgelegt, im Script die einzelnen Seiten unter Rückgriff auf die im Prologue vorgenommenen Einstellungen aufgebaut. Der Prologue kann häufig für viele verschiedene Scripts konstant programmiert und in einer Datei vorgefertigt bereitgehalten werden.

Weiter wird empfohlen, in einem mehrseitigen Dokument die Seiten unabhängig voneinander zu programmieren und nur auf Elemente des Vorspanns zurückzugreifen. Zur Isolation der einzelnen Seiten und zu ihrem Schutz vor ungewünschten Seiteneffekten aus anderen Seiten werden die begrenzenden Operatoren `save` und `restore` bereitgestellt.

Für die statische Codeanalyse — manuell oder maschinell durchgeführt — besonders hilfreich ist eine optisch erkennbare Strukturierung. In PostScript wird dies durch ein erweitertes Kommentierungskonzept realisiert. Während einfache Kommentare durch die Marke *(„Tag")* „%" identifiziert werden und an beliebiger Position stehen können, beginnen *Strukturierungskommentare* immer am Zeilenanfang und führen als *Tag* „%%" oder „%!". Ein Strukturierungskommentar endet in jedem Fall am nächsten Zeilenende.

Neben dem Versionsidentifikator „%!" werden drei Gruppen unterschieden:

- *Header–Comments* beginnen sofort nach der Versionsangabe und enden entweder an der ersten Zeile, die nicht mit „%%" oder „%!" beginnt, oder an einem expliziten „%%EndComments".
- *Body–Comments* dienen vornehmlich zur Gliederung des Scripts, insbesondere zur Einrahmung der einzelnen Seiten des Dokuments.
- *Trailer–Comments* dienen dazu, Informationen, deren Erbringung im Header–Teil mit dem Metastring „(atend)" verzögert wurde, nachzuliefern.

Sowohl im Header– als auch im Trailer–Teil ist die Reihenfolge der Comments nicht signifikant, sofern nicht das gleiche Tag mehrfach auftritt. Bei mehrmaligem Auftreten eines Tags mit sich ausschließenden Werten zählt der jeweils letzte.

Die verfügbaren Strukturkommentare sind im Einzelnen:

- **`%!‹text›` — [Versionsidentifikator] ☑**
  Dieser Kommentar steht immer als erste Zeile eines PostScript–Programmes. Das Bitmuster dieser beiden ASCII–Zeichen (00100101 00100001) dient z. B. im Betriebssystem UNIX zur Identifikation der Datei als PostScript–Text. Nachfolgend bezeichnet ein Klartext, etwa „`PS-Adobe-2.0`", daß der anschließende Code konform zu den Strukturierungsregeln der angegebenen Version ist. Dabei zeigen die ersten 11 Byte („`%!PS-Adobe-`") eine minimale Konformität an, die anschließenden Bytes bis zum Zeilenende ggfs. eine strenge Konformität zu einer bestimmten *Version (PostScript–Level).*

[14] Professor Dr. Niklaus Wirth, ETH Zürich, „Erfinder" des „strukturierten Programmierens" und Entwickler der Sprachen „PASCAL", „Modula", ...

- ❏ %%DocumentFonts: [‹font$_1$› [‹font$_2$›][…]]|(atend) – [Header, Trailer] ☑
  gibt die PostScript–Namen der im Dokument eingesetzten Schriftschnitte an. (Bei vielen Druckern müssen sie vor dem Start des Ausdrucks in einen speziellen Schriftartenspeicher geladen werden.)
- ❏ %%Title: ‹text› – [Header]
  Ermöglicht, einen identifizierenden Freitext als Titel dem Programm beizugeben. Häufig findet hier die generierende Anwendung und der zugrundeliegende Dateibezeichner Eingang.
- ❏ %%Creator: ‹text› – [Header]
  Der Name der Person oder des Programms, etwa des Druckertreibers, die den PostScript–Code erstellt haben.
- ❏ %%CreationDate: ‹text› – [Header]
  System–Datum und –Zeit bei der Erstellung des Codes.
- ❏ %%For: ‹text› – [Header]
  Ein möglicher Empfänger des Dokumentes in Klartext.
- ❏ **%%Pages: ‹pages›|(atend) – [Header, Trailer]**
  gibt die Anzahl der Seiten im gesamten Dokument an. Sie bestimmt sich aus der Anzahl der ausgeführten showpage– oder copypage–Operationen im Programm und ist eine nicht–negative ganze Zahl. Der Wert 0 ist für PostScript–Programme gedacht, die – etwa als Abbildung – in eine andere Seite eingebunden werden.
- ❏ %%BoundingBox: ‹ll$_x$› ‹ll$_y$› ‹ur$_x$› ‹ur$_y$›|(atend) – [Header, Trailer]
  gibt in ganzen Zahlen Koordinaten für die Ausdehnung der gesamten, produzierten Grafik auf der virtuellen Seite an. Dies ist sinnvoll bei einseitigen Dokumenten, die auf eine Seite eines anderen Dokuments eingebunden werden, hat für mehrseitige Dokumente jedoch eher Nachteile und sollte dann ausgelassen werden.
- ❏ %%EndComments – [Header]
  schließt explizit den Header–Part des Programms ab.
- ❏ %%EndProlog – [Body] G
  schließt den Vorspann des Programms ab und kennzeichnet den Beginn des Scripts.
- ❏ **%%Page: ‹label› ‹ordinal› – [Body]** ☑
  kennzeichnet den Beginn einer neuen Seitenbeschreibung und schließt damit eine ggfs. vorhergehende ab. Die Werte „label" und „ordinal" identifizieren die Seite.

‹label› ist ein String, der keine „white Spaces“[15] enthalten darf und im allgemeinen der gedruckten Seitennummer entspricht (etwa ‚xxiii‘ oder ‚III–23‘, ...), ‹ordinal› ist eine positive ganze Zahl, die die Position der Seite in der Folge aller das Dokument ausmachenden Seiten wiedergibt.

- ❏ %%PageFonts: [‹font$_1$› [‹font$_2$›] [...]] – [Body]
  Hier werden die Fonts aus der Liste der *DocumentFonts* aufgeführt, die zur Erstellung der speziellen Seite benötigt werden. Dieser Kommentar muß unmittelbar dem „%%Page“ folgen, und die angegebenen Fonts müssen eine Untermenge der *DocumentFonts* sein. Die Verwendung des Kommentars ist besonders bei speicheraufwendigen, komplexen Seiten empfehlenswert, bei deren Erstellung eventuell Druckerspeicher durch Verdrängung nicht benötigter Fonts gespart werden kann. Ist kein Kommentar „%%PageFonts“ angegeben, werden alle *DocumentFonts* für die Seite verfügbar gehalten.

- ❏ %%Trailer – [Body] ☑
  kennzeichnet das Ende der letzten Seite eines Dokumentes und den Beginn des Trailer–Abschnitts. Alle Anweisungen, die diesem Kommentar folgen, werden nicht auf eine spezielle Seite sondern auf das gesamte Dokument bezogen; es können beispielsweise aufräumende Vorgänge hier gestartet werden.

Für die vorliegende Studienarbeit sind die Strukturierungskommentare der PostScript–Programme von besonderer Bedeutung. Der vorzustellende Link–Navigator und der Link–Editor bauen auf ihnen auf.

[15] als „white Spaces“ (englisch – „weiße Zwischenräume“) bezeichnet man solche Zeichen, die im Druckbild gewollt ein Stück weißes, unbedrucktes Papier ergeben. In PostScript sind die „white Spaces“ das Leerzeichen *SP* (ASCII $32_{10}$), der Horizontaltabulator *HT* (ASCII $9_{10}$) und die „Newline“–Sequenz, also das Zeilenende *LF* (ASCII $10_{10}$) oder Wagenrücklauf *CR* (ASCII $13_{10}$) und *LF*.

# 4 Die Navigationskomponente

## 4.1 Anbindung an die Prozeßmodellierung

Bei der Realisierung der vorliegenden Arbeit ist als wesentliche Randbedingung die Anbindung des Hypertextsystems an die bestehende Prozeßmodellierung vorgegeben.

Die Navigationskomponente muß deswegen aus der Prozeßmodellierung heraus nicht nur gestartet, sondern auch je nach dem startenden Prozeßschritt zu einem bestimmten Knoten im Hypertextnetz positioniert werden können.

In der informationstechnischen Modellierung werden die Attribute der Prozeßschritte in sogenannten *Slots, Schlitzen,* untergebracht. Üblicherweise sind es numerische Werte wie Temperaturen oder textuelle Werte wie die Bezeichnungen der benötigten Ausrüstungsgegenstände, die in diesen Slots abgespeichert werden, also Fakten zu dem modellierten Prozeßschritt selbst. Diese Slots können jedoch auch zur Unterbringung eines Startkommandos für den Hypertext–Navigator genutzt werden. Es kann dabei sinnvoll sein, in einer entsprechenden (hohen) Objektklasse einen Slot auszuzeichnen, der dann in jeder Objektinstanz den Eintritt in das Hypertextnetz erlaubt, auf dem Wege der Vererbung auch in Instanzen verfeinerter Objektklassen. Dies erfordert auf Seiten des Hypertextnetzes allein das Vorhandensein einer Hypertextseite, die standardmäßig angesprungen werden muß, wenn kein ausgezeichneter Netzknoten der aufrufenden Instanz zugeordnet ist.

Die Zuordnung von Hypertextseite zu Objektinstanz geschieht während der Ausführung der Prozeßmodellierung über das Editieren von Prozeßattributen. Durch den Eintrag des Start–Kommandos in einem Slot wird das Kommando selbst zum (textuellen) Attribut des aufrufenden Prozeßschrittes.

Wird in dem entsprechenden *Slot* des Prozeßschrittes also ein textueller Aufruf untergebracht, erfordert dies, daß die Navigationskomponente durch ein übliches *Shell–Kommando*[16] startbar und über entsprechende *Aufrufparameter* positionierbar sein muß.

Eine zu Beginn der Studienarbeit vorgenommene Untersuchung von möglichst unter GNU–Lizenz oder ähnlich frei verfügbarer Software lieferte als Ergebnis, daß kein fertiges, als Paket befriedigend einsetzbares Produkt existiert. Die Suche wurde über das

[16] UNIX ist ein nach dem Schichtprinzip aufgebautes Betriebssystem, das die Hardware vor direkten Zugriffen kapselt. In der äußersten Schicht oder Schale, englisch „shell", werden die Kommandos des Bedieners abgesetzt und durch die Schichten aufbereitet und schließlich an den Systemkern zur Verarbeitung weitergereicht.

weltweite Forum der Internet–News[17], die eine eigene Hypertext–Newsgruppe unterhält, und eine Anfrage bei der Sun Microsystems GmbH durchgeführt. Die auf diesem Wege gefundenen Produkte arbeiten entweder mit eigenen Dokumentformaten, die die maschinelle Übernahme vorhandener Schriftstücke vereiteln und zur kompletten Neuerfassung des zu verarbeitenden informellen Materials zwingen, oder sie sind nur selbständig ausführbar und sperren sich dem textuellen Aufruf und der Positionierung aus der Prozeßmodellierung heraus.

Die Entscheidung zugunsten des Dokumentformates PostScript ermöglichte jedoch, im eingangs vorgestellten Konzept die Navigation auf Basis des „HelpViewer" von Sun Soft Inc., einer Tochter der Sun Microsystems Inc., zu realisieren.

## 4.2 Die „Viewer"–Applikationen in OpenWindows 3.0

Sun liefert mit seinen UNIX–Betriebssystemen „Sun OS 4.1.x" und „Solaris" für die Sun–4–Architekturen das Windows–System „OpenWindows" in der aktuellen Version 3.0 aus. In diesem System sind neben dem bekannten Window–Manager „OpenLook" mehrere Desktop–Tools enthalten, die noch auf das PostScript–basierte „NeWS" zurückreichen, darunter auch die Anwendungen „PageView" und „HelpViewer".

- ❏ *„PageView"* ist ein Dienstprogramm zur Interpretation von PostScript–Code und zur Anzeige seiner Ausgabe (seitenweise) in einem Fenster des Windows–Systems.

  Dabei können über *Buttons*[18] im Randbereich des PageView–Fensters *Pop–Up–Menüs*[19] aktiviert werden, die den Zugriff auf das Dateisystem zum Laden und Drucken von Quellcode–Dateien ermöglichen, das Blättern um eine Seite nach vorn oder zurück und zur ersten bzw. letzten Seite des Dokuments steuern, die Gestalt des Fensters und Festlegung der Ausgabequalität vornehmen und die Betrachtung, manuelle Veränderung und Reinterpretation des Quellcodes in einem eigenständigen Text–Fenster gestatten.

---

[17] „Das Internet" ist der größte weltweite, offene Zusammenschluß von öffentlichen und kommerziellen Netzen. Mit NNTP–News wird hier ein Mehrwertdienst angeboten, der es ermöglicht, in verschiedenen Foren, den „NewsGroups", zu bestimmten Themen mit anderen Benutzern weltweit zu kommunizieren. Eigene Beiträge werden über den lokalen Host in der Newsgroup „gepostet" und über das Netz in Minuten, maximal Stunden an alle teilnehmden Hosts propagiert. Entsprechende Newsreader–Werkzeuge ermöglichen, Newsgroups zu abonnieren („subscribe") und die Artikel darin zu lesen.

[18] Ein Button, zu deutsch Knopf, ist ein grafisch einem Druckknopf nachgebildeter Bereich auf einem Rasterbildschirm. Wird er mit einem optischen Zeigegerät (Maus) aktiviert, löst er bestimmte Aktionen des Systems aus.

[19] Als Menü bezeichnet man eine Auswahlliste von möglichen Aktionen oder Einstellungen, die dem Benutzer auf dem Bildschirm angeboten werden. Je nach der Art des Einblendens des Menüs auf den Bildschirm unterscheidet man Pop–Up– (erscheinen „aus dem Nichts" und „schweben über dem Bildschirm"), Drop–Down– („klappen aus dem auslösenden Objekt (Button) herab") und Pull–Down–Menüs (wie Drop–Down–Menüs, allerdings nur, solange das auslösende Ereignis („Mausknopf betätigt") anhält).

Abbildung 4 zeigt eine im PageView–Fenster angezeigte PostScript–Seite, die zur Verfügung stehenden Menüs und das mit dem View gekoppelte PostScript–Programm.

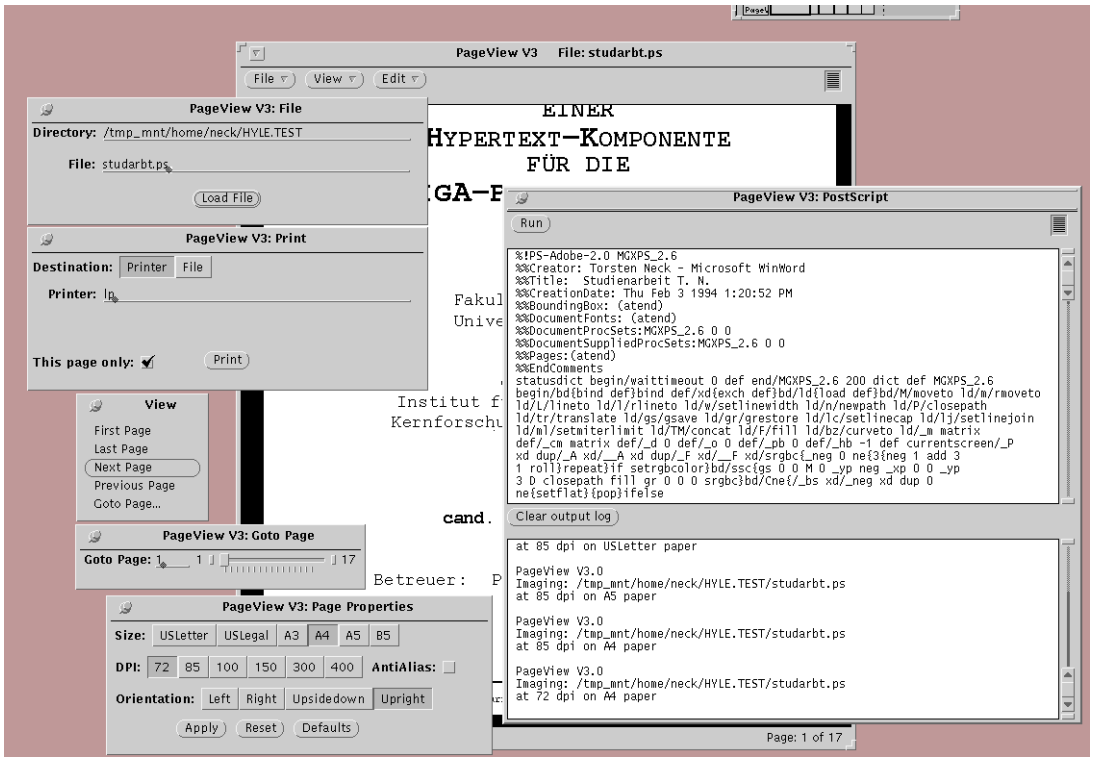


*Abb. 4: Erscheinungsbild des PageView*

❏ *„HelpViewer"* ist eine Erweiterung des „PageView". Neben dem reinen *„View"* der PostScript–Ausgabe, der Anzeige auf dem Bildschirm, steht hier ein kompletter, leistungsfähiger Hypertext–Navigator zur Verfügung. Dafür besteht die Möglichkeit der Code–Manipulation in einem separaten Text–Fenster und der Reinterpretation nach Änderungen im HelpViewer nicht.

## 4.3 Bedienung des HelpViewer

Die Steuerung des Navigators geschieht — abgesehen von seinem Start durch einen textuellen Shell–Aufruf — grafisch intuitiv mit einem optischen Zeigegerät, im Normalfall der Maus, oder über besondere Tasten auf der Tastatur.

Abbildung 5 zeigt den Abdruck eines HelpViewer–Fensters. Es ist für eine einfache grafische Steuerung in vier funktionale Bereiche gegliedert:

❏ Der Rahmen und die Titelzeile des Applikationsfensters stellen die in Window–Managern üblichen Aktionen auf die Form des Fensters an sich zur Verfügung, in der abgedruckten Darstellung speziell im Erscheinungsbild des „Open–Look".

❏ In der Statuszeile am unteren Rand des Applikationsfensters werden Informationen des PostScript–Interpreters und des Viewers ausgegeben. Wie im abgedruckten Beispiel ersichtlich, sind dies im Normalfall der Titel des Dokumentes aus dem Strukturierungskommentar „`%%Title`" und die aktuelle Vergrößerungsstufe der Ansicht. Hier können jedoch auch Fehlermeldungen erscheinen, wenn z. B. die geladene Datei sich nicht durch die ersten zwei Byte „%!" als PostScript identifiziert, oder tatsächliche Statusanzeigen während des Interpretiervorgangs.

Der überwiegende Teil der Meldungen von HelpViewer wird allerdings über *stdout* direkt an die Konsole oder das Sun–Konsolfenster „*contool*" der Workstation ausgegeben, nicht an das aufrufende Shell–Fenster.

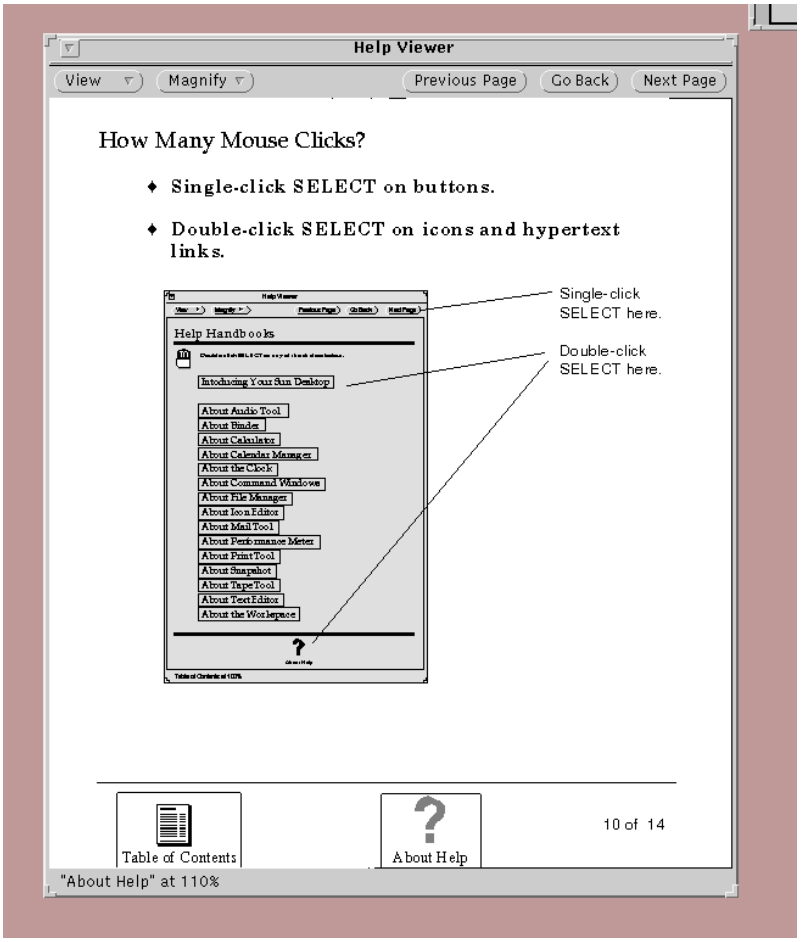


*Abb. 5: Ein HelpViewer–Fenster auf dem Open–Look Workspace*

❏ Direkt unter der Titelzeile befindet sich eine Knopfzeile zur Steuerung des HelpViewer über Pull–Down–Menüs (Abbildung 6).

Die beiden Buttons im linken Teil dienen zur Anpassung der Gestalt des angezeigten Dokumentes: im Menü „View" kann beispielsweise eingestellt werden, ob Hypertext–Links in der Ansicht hervorgehoben werden sollen (wie in Abbildung 5 am

unteren Rand bei den Icons[20] „Table of Contents“ und „About Help“ oder in Abbildung 6 bei den Zeilen „Getting Help“, „Quitting Help“ und „Navigating Through Help“) oder nicht. In „Magnify“ können verschiedene Vergrößerungsstufen des Fensters und der darin enthaltenen Ansicht gewählt werden.

Die drei Buttons im rechten Teil sind unterstützend für die Hypertext–Navigation vorgesehen:

Mit „Next Page“ (Taste «PgUp» [R9] auf einer Sun Typ–4–Tastatur) blättert man in der natürlichen Reihenfolge der Dokumentseiten weiter, bzw. mit „Previous Page“ («PgDn» [R15]) um eine Seite zurück, die Existenz jeweils vorausgesetzt.

Der Button „Go Back“ («Undo» [L4]) schließlich dient zum schrittweise Rückverfolgen eines durch Links und/oder Blättern eingeschlagenen individuellen Pfades.

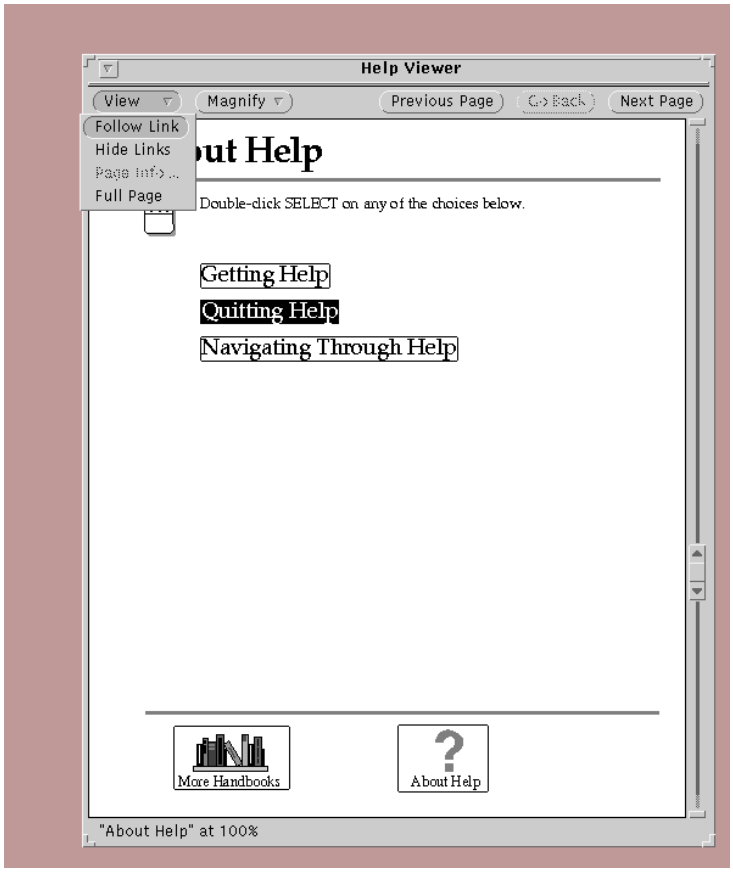


*Abb. 6: Ein selektierter Link und seine Aktivierung im View–Menü des HelpViewer*

❑ Den größten Teil des Fensters nimmt der View–Bereich für die PostScript–Ausgabe ein. Er ist im HelpViewer nicht nur Ausgabemedium sondern wird insbesondere interaktiv zur Auslösung von Link–Bewegungen eingesetzt.

[20] Icons, deutsch seltener Ikonen oder Pictogramme, nennt man die (meist mit einem Etikett versehenen) Sinnbilder, hinter denen sich Anwendungen, Aktionen oder Objekte verbergen, die durch Anklicken der Icon mit der Maus oder einem anderen Zeigegerät aktiviert werden können.

Der Start des HelpViewer geschieht entweder aus dem „Workspace"–Menü des Open–Look–Windowmanagers heraus durch Mausklick und führt dann üblicherweise in ein Hypertext–Netz von Handbüchern des Herstellers Sun zu den Werkzeugen des Systems OpenWindows 3.0, oder aber in der hier relevanten Variante als textueller Aufruf aus einer UNIX–Shell heraus.

Die Umgebungsvariable „$HELPPATH" des Betriebssystems sollte die Namen der Verzeichnisse enthalten, in denen die Dateien für das Hypertext–Netz abgespeichert sind, sie müssen sich sonst in dem zum Zeitpunkt des Aufrufs „aktuellen" Verzeichnis („$cwd") befinden. Ebenso sollte die Umgebungsvariable „$OPENWINHOME" richtig gesetzt sein, damit der Navigator gestartet werden kann. Die Werkzeuge von Open Windows 3.0 befinden sich bei üblicher Installation des Dateisystems im Katalog „/usr/openwin/bin".

Bestehen diese Voraussetzungen, wird der Navigator durch das Kommando

```
helpviewer ‹help-info-file›
```

gestartet. Dabei ist die Angabe des Parameters ‹help–info–file› notwendig („mandatory"), da in der Oberfläche des HelpViewer keine Möglichkeit zum Zugriff auf das Dateisystem besteht, außer implizit über die Verfolgung von Interlinks.

Es ist möglich und empfehlenswert, den HelpViewer durch das Kommando

```
helpviewer ‹help-info-file› &
```

als selbständigen (asynchronen) Hintergrundprozeß zu starten, da so die Shell sofort einen neuen Prompt ausgibt und in der Lage ist, weitere Kommandos entgegenzunehmen und auszuführen.

Die Beendigung des Navigator–Prozesses geschieht über die Option „Quit" im Applikations–Menü des Windowmanagers (Anklicken des HelpViewer–Rahmens oder der HelpViewer–Titelzeile mit der rechten Maustaste „Menu").

## 4.4 Der Link–Mechanismus und die zugrundeliegende Verarbeitungsstruktur

Ist die Anzeigeoption für Links im „View"–Menü gesetzt, werden sie als Rechtecke um die als Ausgangspunkte dienenden Seitenabschnitte im Viewbereich angezeigt.

Ein „Doppel–Klick" mit der linken der bei Sun üblichen drei Maustasten („Select") aktiviert den Sprung des Links, in dessen abgeschlossenem Rechteck auf der Anzeige sich der Mauscursor im Augenblick des Vorfalles befindet.

HelpViewer erlaubt, als Ausgangspunkt eines Links jeden beliebigen, rechteckigen Ausschnitt einer Seite festzulegen, und ist nicht an bestimmte dargestellte Objekte gebunden. Die Auslösung erfolgt allein durch den Vorfall des Doppelklicks und das Zutreffen der Inzidenzbedingung der augenblicklichen Mauskoordinaten auf der Fläche eines Link–Rechtecks. Dabei ist unerheblich, ob im „View"–Menü die Hervorhebung der Startpunkte ein– oder abgeschaltet ist. Es ist daher möglich, einzelne Schriftzeichen, Wörter,

Grafiken, Kombinationen daraus, bis hin zur kompletten Seite als Startstellen zu definieren.

Die zugrundeliegende Verwaltung legt als steuernde Datenstruktur einen speziellen Stack an. Auf ihm werden die Dateinamen und Seitennummern aller besuchten Seiten in der Reihenfolge des Besuchs abgelegt. Auf diese Weise kann durch Ansteuern der Top–Adresse und anschließendes Pop auf den Link–Stack die Rückverfolgung über den „Go Back"–Button realisiert werden, da die LIFO–Struktur die Seiten in umgekehrter Reihenfolge des Besuchs zurückliefert.

HelpViewer kann mehrfach gestartet werden, selbst auf dem gleichen Hypertextnetz. Jeder Instanz wird dann ein eigener Prozeß mit einem eigenen, unabhängigen Navigations–Stack zugeordnet. Pro Navigationsprozeß wird jedoch genau *ein* Stack der besuchten Seiten *flüchtig* geführt, also bei der Terminierung des Prozesses wieder gelöscht. Es ist deshalb weder möglich, nach (partieller) Rückverfolgung eines Pfades diesen automatisch wieder aufzunehmen, noch können beendete Navigationssitzungen unter Rückgriff auf einen abgespeicherten, schon beschrittenen Pfad (Konzept der „Lesezeichen") weiterverfolgt werden.

## 4.5 Syntax und Semantik der Link–Kommentare

Zur Definition von Links in einem für HelpViewer zu verwendenden PostScript–Programm wird der Mechanismus der Strukturierungskommentare um ein Tag erweitert. Diese Erweiterung ist firmenspezifisch, von Sun Soft Inc. entwickelt, und darf keinesfalls als Bestandteil von PostScript verstanden werden. Da es sich aus Sicht der PostScript–Syntax aber um Kommentare handelt, ist die Verträglichkeit mit allen existierenden und zukünftigen Levels sichergestellt.

Die Link–Erweiterung bleibt ausdrücklich sowohl in der Syntax als auch in der Semantik undokumentiert. Die einzige Information, die von Sun Soft dazu vorliegt, ist der Man–Page[21] zu HelpViewer zu entnehmen: „HelpViewer has a simple hypertext mechanism. Documents may have hypertext links to other handbooks or to other pages in the same handbook. ... HelpViewer hypertext links can also be used to initiate system processes, enabling an author to launch shell scripts and applications or commmunicate with other processes directly from a link in a document. ... The HelpViewer hypertext link format is not documented." Zu deutsch: „HelpViewer besitzt einen einfachen Hypertext–Mechanismus. Dokumente können Links zu anderen Handbüchern oder zu weiteren Seiten im selben Dokument aufweisen. ... Die Hypertext–Links des HelpViewer können auch dazu dienen, Systemprozesse anzustoßen, einem Autor das Starten von Shell–Scripts und Applikationen zu ermöglichen oder mit anderen Prozessen direkt aus einem Link heraus

[21] Allen UNIX–Kommandos und den meisten Applikationen sind kurze Beschreibungen und Hilfestellungen zugeordnet, die auf dem Bildschirm als quasi elektronisches Handbuch angezeigt werden können. Sie werden *Manual–Pages* oder kurz *Man–Pages* genannt und mit dem Kommando „man" angezeigt.

zu kommunizieren. ... Das Format der Hypertext–Links in HelpViewer ist nicht dokumentiert."

Es war also notwendig, Syntax und Semantik der Links durch eigene Untersuchungen zu klären.

Ein Link hat die folgende Gestalt:

```
%%Link:v1:0:91 420 272 440:view::2
```

oder:

```
%%Link:v1:0:69 3 164 57:view:handbooks/top.toc.handbook:1
```

Die Bestandteile des Eintrags einer Hypertextverbindung werden jeweils durch genau ein Kolon „:" separiert und sind im einzelnen:

- **`%%Link`**
  das zu den Dokument–Strukturierungs–Kommentaren hinzugekommene Sun–proprietäre Tag.
- `v1`
  die Funktion dieses Parameters konnte nicht ermittelt werden.
- `0|1`
  bestimmt die Wirkungsweise der Selektion des Links. Bei dem von Sun selbst ausschließlich eingesetzten Wert 0 und jeder geraden natürlichen Zahl an dieser Stelle ergibt sich das früher beschriebene Verhalten, daß ein Link durch Doppelklick aktiviert wird, ein einfaches Anklicken bewirkt seine Hervorhebung durch farbliche Invertierung des begrenzenden Rechtecks, aber keine Aktivierung. Der Wert 1 oder jede ungerade natürliche Zahl an dieser Stelle ändert das Verhalten derart, daß schon ein einfaches Anklicken des Links diesen auch aktiviert.
- **‹$ll_x$› ‹$ll_y$› ‹$ur_x$› ‹$ur_y$›**
  Dieser Parameter hat vier ganzzahlige Bestandteile, jeweils durch ein „white Space" (ausgenommen „Newline", das die Kommentarzeile vorzeitig abschließen würde) separiert. Es handelt sich um die Koordinaten des begrenzenden Rechtecks auf der virtuellen Ausgabeseite des PostScript–Programmes in der Form, die bereits im Programmbeispiel „LRE" in Abschnitt 3.2.3 dieser Arbeit verwendet wurde: zunächst die horizontale Koordinate der unteren Begrenzung, die vertikale Koordinate der linken Grenze, dann die horizontale Koordinate der oberen Grenze und die vertikale Koordinate der rechten Begrenzung.
- `view|‹string›`
  Die Funktion dieses Parameters konnte nicht genau ermittelt werden. Festgestellt werden konnte, daß das von Sun selbst hier ausschließlich eingesetzte „view" oder ein beliebiger String, der keine „white Spaces" enthält, bei fehlerhaften Angaben in den anderen Parametern als erster Bestandteil der auf der Konsole ausgegebenen Fehlermeldung erscheint.
- **`[[‹path›]‹handbook_file›]`**
  An dieser Stelle wird das Zieldokument des Hypertext–Links angegeben. Bleibt dieser Parameter leer, so handelt es sich um einen Intralink, andernfalls um einen

Interlink zu der eventuell durch eine volle Pfadangabe ergänzten Zieldokumentdatei.

- ❑ **‹page›**
  gibt als letzter, notwendiger Bestandteil die Zielseite des Links im ausgewählten Zieldokument an. Die angegebene Seite muß im Zieldokument existieren und *konform* mit dem Strukturierungskommentar %%Page versehen sein. Die hier als Sprungziel verwendete Seitennummer wird bei Numerierung in der Form „‹label› ‹ordinal›“ mit dem Wert für ‹ordinal›, nicht für ‹label› verglichen (vgl. Abschnitt 3.3).

Die in dieser Weise aufgebauten Link–Anweisungen können an beliebiger Stelle im Script der Seite angegeben werden, auf der die Hypertextverbindung starten soll. Die Auszeichnung im View erfolgt beim Aufbau der Seite in der Reihenfolge des Auftretens der „%%Link“–Zeilen. Doch auch hier greift die für PostScript charakteristische LIFO–Struktur der Verarbeitung: es besteht die Möglichkeit, optisch geschachtelte Links auf einer Seite zu definieren. Dabei ist darauf zu achten, daß ein kleineres Rechteck, das echt in einem zweiten, größeren enthalten ist, nur dann aktivierbar ist, wenn es im Script nach dem umgebenden, größeren definiert wird. Die LIFO–Struktur legt die später definierten Startpositionen *auf* die früher angegebenen und kann auf diese Weise in ungünstigen Fällen zu Verdeckungen führen.

# 5 Der Link–Editor

## 5.1 GhostScript und GhostView

### 5.1.1 GhostScript

Im Anbetracht des hohen Aufwandes und der Kosten einer PostScript–Belichtung oder eines Ausdruckes nur zu Kontrollzwecken und insbesondere durch das Konzept von PostScript begünstigt, auf jedem möglichen Ausgabemedium ohne Codeveränderung das bestmögliche Erscheinungsbild des Satzes zu liefern, haben sich die bereits am Beispiel PageView vorgestellten, sogenannten *Viewer* oder *Previewer* weit verbreitet, die einen Software–Interpreter für gängige Rasterschirme zur Verfügung stellen und ihn mit einer meist menügesteuerten, interaktiven Oberfläche versehen, die dem Benutzer ermöglicht, einzelne Seiten anzusteuern und zu blättern.

Eines der am weitesten verbreiteten Produkte im Sektor der PostScript–Previewer ist das unter der GNU–Lizenz vertriebene *GhostScript* der *Aladdin Enterprises Inc.* Neben der kostenlosen Verfügbarkeit ist sicher ein Hauptgrund für die weite Verbreitung von GhostScript, daß eine große Zahl von Treibern in diesem Paket enthalten ist: neben den gebräuchlichen Non–PostScript–Druckern des „PCL–Standards[22]“ und einiger anderer mehr werden die Chipsets vieler Grafikkarten speziell unterstützt, und es können weitverbreitete Grafikdateiformate wie GIF und PCX erzeugt werden.

Darüber hinaus ist GhostScript im Quellcode erhältlich und läßt so unter GNU–Lizenz Anpassungen und Korrekturen durch den Anwender selbst zu.

GhostScript ist im Sprachumfang gegenüber PostScript leicht eingeschränkt, es umfaßt jedoch den kompletten Level 1 von PostScript. Die Abweichungen werden in einem mitgelieferten „README–File“ beschrieben, das im Anhang unter Abschnitt 7.3 in Auszügen abgedruckt ist.

### 5.1.2 GhostView

Zu dem interpretativen GhostScript existiert ein menügesteuerter Aufsatz, „Add–On“, unter *X11*, der zwar den Zugang über einzeln eingetippte GhostScript–Anweisungen verschließt, dafür jedoch eine einfach zu handhabende Benutzeroberfläche mit Menüzugriff auf das Dateisystem bietet. Das Add–On nennt sich *„GhostView“* und wurde

[22] PCL steht für „Printer Control Language“, zu deutsch Druckersteuersprache, und ist ein Warenzeichen von Hewlett Packard. PCL hat sich neben PostScript als De–facto–Standard vor allem bei der semiprofessionellen Druckersteuerung eingebürgert.

überwiegend an der University of Wisconsin–Madison, USA, entwickelt und ebenfalls unter der GNU–Lizenz verbreitet. Abbildung 7 zeigt das Erscheinungsbild von GhostView auf dem Bildschirm.

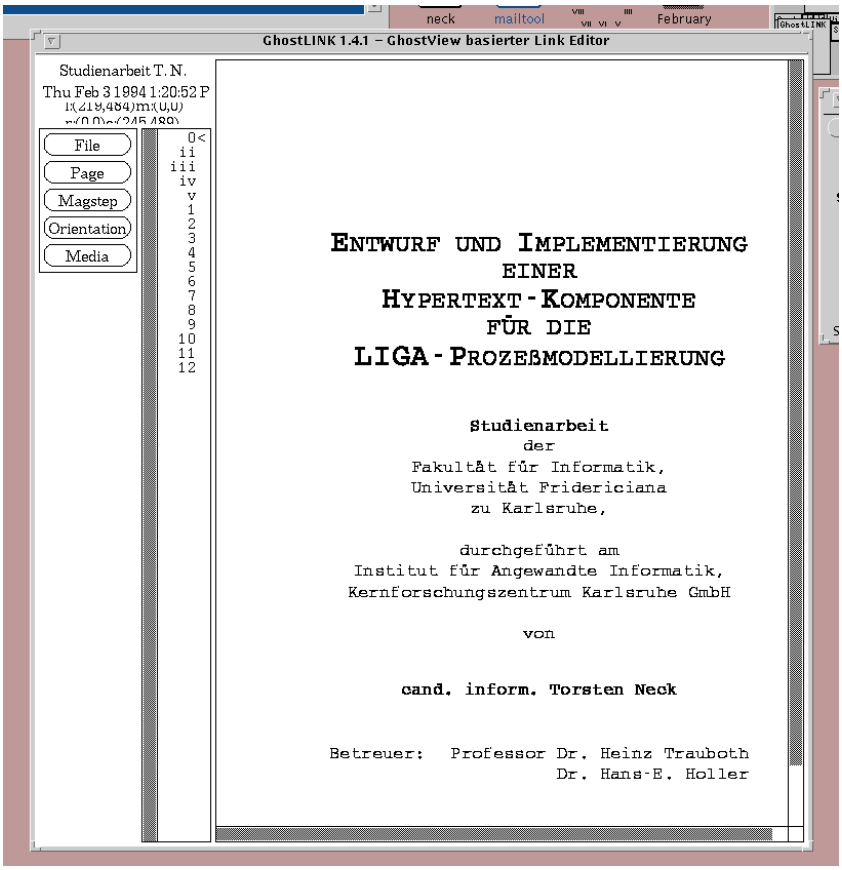


*Abb. 7: Erscheinungsbild des GhostView auf dem Workspace*

Die Bedienung geschieht ähnlich wie bei dem bereits früher kurz vorgestellten PageView:

In einer Button–Leiste im linken Randbereich des Applikationsfensters stehen fünf Menüs, „File", „Page", „Magstep", „Orientation" und „Media", zur Verfügung, um GhostView zu handhaben.

- ❏ Das Menü „File" bietet Optionen zum Öffnen von GhostScript–Programmdateien, zum Drucken von Dokumenten und ausgewählten Seiten, zum Abspeichern ausgewählter Seiten und zum Verlassen der Applikation.
- ❏ „Page" ermöglicht das Blättern der Seiten und ihre Markierung z. B. für den späteren Druck.
- ❏ In „Magstep" wird ein Zoom der Ansicht in 11 Stufen von –5 bis +5 ermöglicht. Dabei bedeuten der Wert 0 die Ansicht in Originalgröße, negative Werte ein Verkleinerung und positive Werte eine Vergrößerung der Ansicht.
- ❏ „Orientation" dient zur Drehung der Koordinatenachsen in 90 °–Schritten oder ihrer Spiegelung.

- In „Media“ schließlich kann ein Verhältnis von horizontaler zu vertikaler Seite aus einer Liste von Standardformaten festgelegt werden. Allen voran stehen die amerikanischen Papierformate, gefolgt von den DIN–Formaten A3, A4, A5, B4 und B5 und einigen Sonderformaten[23]. Diese Formate können der Ansicht zugewiesen werden und überschreiben dann die aus dem Quellcode abgelesenen Formateinstellungen der angezeigten Seite.

Neben der Button–Leiste findet sich eine scrollbare Liste mit allen aus den `‹label›`–Parametern der „%%Page“–Strukturkommentare gewonnenen Seitennummern. Die jeweils angezeigte, aktuelle Seite ist in dieser Liste durch ein nachgestelltes „<“–Zeichen markiert. Eine ausgewählte Seite kann mit Hilfe dieser Liste schnell angezeigt werden, indem die Seitennummer mit der mittleren Maustaste angeklickt wird.

GhostView bezieht alle notwendigen Daten, wie z. B. die Seitenlabels oder das Seitenformat und die Seitenausrichtung, aus entsprechenden Strukturkommentaren, erfordert also zur störungsfreien Anzeige einen konformen Quellcode. Die von GhostView — nicht GhostScript — ausgewerteten Strukturkommentare sind in Anhang unter Abschnitt 7.4 verzeichnet.

## 5.2 Die grundlegende Programm–Struktur in GhostView

GhostView ist in der Sprache C programmiert und verwendet als X11–Applikation intensiv die Programmierbibliotheken „X–Toolkit“ („Xt“) und „X–Athena–Widgets“ („Xaw“). Ohne auf Einzelheiten der Programmierung einzugehen, sollen die Modulstruktur und die Charakteristika der Bibliotheken nachfolgend knapp beschrieben werden.

### 5.2.1 Die wesentliche Modulstruktur von GhostView

Der Quellcode von GhostView verteilt sich auf vierzehn Dateien mit einem Umfang von 180 DIN–A4–Seiten im Ausdruck. Die Dateistruktur entspricht weitgehend der Modulstruktur. Da eine Änderung des Codes zwar gestattet, jedoch nicht vorgesehen ist, liegt keine Dokumentation (Modulführer) vor, und die Kommentierung im Quellcode ist spärlich gehalten.

Die Moduln sind im einzelnen:

- „main.c“ — der Basismodul, der das Applikationsfenster aufbaut und die Dienste koordiniert.
- „misc.c“ — kontrolliert alle Vorgänge, die nicht von „callbacks.c“ und „action.c“ ausgeführt werden, z. B. den Aufbau des Inhaltsverzeichnisses (Seitenliste) oder die Anzeige von Titel und Datum des PostScript–Programms.

[23] Es werden die folgenden Formate angeboten, ihre Ausdehnung ist in Punkt in Klammern nach der Bezeichnung in der Form („horizontal“·„vertikal“) angegeben: Letter (612·792), Tabloid (792·1224), Ledger (1224·792), Legal (612·1008), Statement (396·612), Executive (540·720), A3 (842·1190), A4 (595·842), A5 (420·595), B4 (729·1032), B5 (516·729), Folio (612·936), Quarto (610·780), 10x14 (720·1008).

- **„callbacks.c"** – definiert die Standardroutinen, die den Aufrufen und Ereignissen (*Events*) zugeordnet sind.
- „actions.c" – hier sind die in den Menüs angebotenen Dienste programmiert.
- **„dialogs.c"** – stellt eine Standard–Dialogbox für GhostView bereit.
- „Ghostview.c" – hier ist die Hauptfunktionalität von GhostView programmiert, insbesondere die Kommunikation und Kooperation mit dem zugrundeliegenden GhostScript.
- **„ps.c"** – führt das vorbereitende Scannen des PostScript–Programmes durch und extrahiert u. a. die in Anhang 7.4 aufgeführten, ausgewerteten Strukturkommentare.
- „getenv.c", „setenv.c", „strcasecmp.c" – sind Hilfsmoduln für die Einbettung von GhostView in das UNIX–Betriebssystem.
- „SelFile.c" – gestaltet die Oberfläche für den Zugriff auf das Dateisystem beim Öffnen oder Speichern von Dateien und bedient sich dabei der Moduln
- „Dir.c", „Path.c" und „Draw.c", die die entsprechenden Dienste bereitstellen.

### 5.2.2 Eingesetzte Programmierwerkzeuge für X11

Die Programmierung einer grafischen Benutzeroberfläche wie X11 erfordert immer wiederkehrende Konstrukte für die vielfältigsten Standardanwendungen, etwa *Buttons*, *Fenster* mit *Titelzeilen*, *Scrollbars*, *Dialogformularen* mit *Textfeldern* und Buttons u. v. m. Es wäre eine sehr mühsame und für den Programmierer unzumutbare Arbeit, für jede Applikation stets unter Verwendung reiner Funktionen der X11–Standardbibliothek „Xlib" diese Objekte zu programmieren. Deswegen werden sogenannte *„Werkzeugkästen"* oder *„Toolkits"* angeboten, die das Erstellen von Dialogschnittstellen erleichtern. Solche Toolkits enthalten üblicherweise eine Benachrichtigungskomponente (notifier, input loop, event dispatcher), die für jedes *Ereignis (Event)* die entsprechende Prozedur aufruft, eine Sammlung von vorgefertigten Fenstertypen (Menüs, Scrollbars, Schalter, Dialogkästen, ...), eine Kompositionsmethode, mit der ein Entwickler die vorgefertigten Fenstertypen zusammensetzen kann, eine Verfeinerungsmethode, mit der sich die vorgefertigten Fenstertypen verfeinern und anpassen lassen, und Prozeduren zur Abfrage von Benutzereinstellungen.

Die gebräuchlichste Kompositionsmethode ist die des *Schachtelungsbaumes*, bei der jedes Fenster außer dem *Grundfenster* genau ein umschließendes *„Elternfenster" („Parent Window")* besitzt, das für die Plazierung und Größe all seiner *„Kinder"* zuständig ist.

Eine weit verbreitete Verfeinerungsmethode ist das Konzept der *Vererbung (Inheritance),* bei dem jeder Fenstertyp als abstrakter Datentyp mit Attributen aufgefaßt wird. Attribute sind hierbei sowohl Datenfelder als auch Operatoren (typspezifische Prozeduren). Die Fenstertypen sind in einer Vererbungshierarchie – häufig einem Baum – angeordnet. Ein Fenstertyp erhält implizit von seinem *Vorfahr,* einem Vorgänger im Baum, dessen Attribute, mit Ausnahme derer, die im aktuellen Fenstertyp umdefiniert werden. Dem

aktuellen Fenstertyp können eigene Attribute hinzugefügt werden. Ähnliches gilt bei den für die Fenstertypen definierten Operatoren. Verschiedene Fenstertypen können Operatoren mit gleichen Namen (und Signaturen[24]) besitzen, die dennoch typspezifisch implementiert sind. Der Anwendungsprogrammierer braucht sich dann jedoch nur ein Konzept zu merken (beispielsweise „erzeugen", „löschen", „vergrößern", „verkleinern").

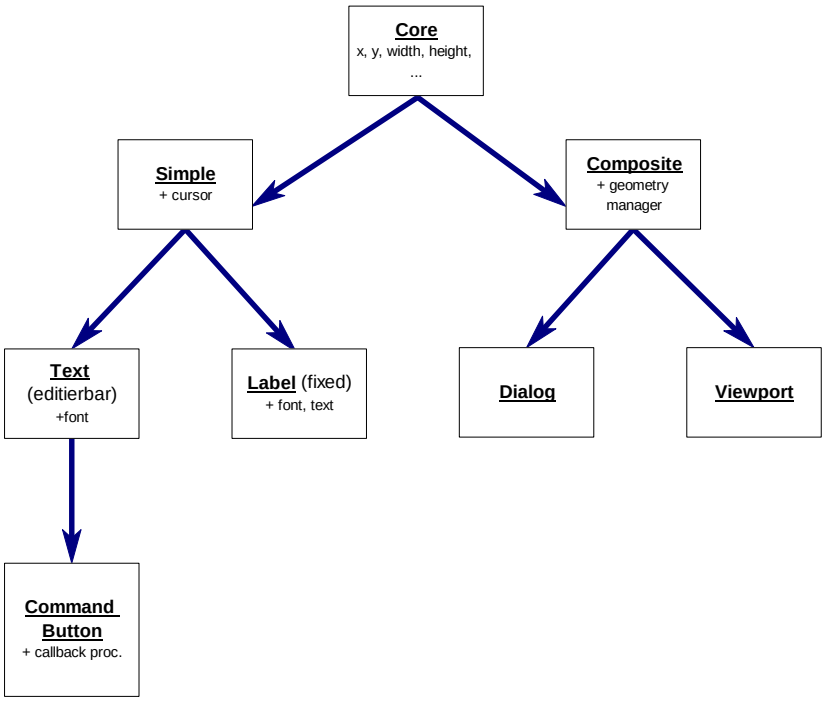


*Abb. 8: Vererbungsbaum des Athena–Widget–Sets*

Ein typischer solcher Werkzeugkasten ist die in GhostView intensiv genutzte „Xt–Library" („Xt" steht dabei für „X–toolkit"). Xt bietet eine Sammlung von *„Widgets"*[25] an. Ein „Widget" ist ein X11–Fenstertyp zusammen mit seiner Ein–/Ausgabe–Semantik. Aus Benutzersicht ist es also ein Fenstertyp, der Information darstellt und auf Benutzereingaben reagiert, aus Sicht eines Anwendungsprogrammierers ein abstrakter Datentyp, dessen Implementierungsdetails verborgen sind und der nur durch angegebene Funktionen und Attribute manipuliert werden kann, und aus Sicht des Widgetprogrammierers ein Fenstertyp in einer Vererbungshierarchie. Xt legt dabei fest, daß die Vererbung *einfach* sein muß, ein Widget also höchstens einen direkten Vorfahr haben darf. Die Vererbungshierarchie ist somit ein Baum.

[24] Die Signatur eines Operators legt fest, welche Operanden von welchem Typ und in welcher Reihenfolge benötigt werden.

[25] Das Wort „widget" stammt aus der amerikanischen Umgangssprache und ist etwa mit „Dingsbums" zu übersetzen.

Eine solche Widget–Hierarchie, die auch in GhostView zum Einsatz kommt, ist die universell verfügbare Athena–Library (Xaw)[26]. Abbildung 8 zeigt diesen stark vereinfachten Vererbungsbaum.

Hierbei sind die einzelnen Widget–Klassen wie folgt zu verstehen: Core–Widgets sind einfache Rechtecke (Attribute sind nur die Rechtecksgeometrie), beim Simple–Widget tritt noch ein Zeiger (Mauscursor) hinzu. Ein Label–Widget ist ein Rechteck um einen Text, in der Verfeinerung zum Command–Button–Widget kommt der Selektionsmechanismus hinzu, so daß ein Button in der Lage ist, eine Callback–Prozedur aufzurufen. Composite–Widgets sind Core–Widgets, die ein oder mehrere andere, eingeschlossene Widgets enthalten (Abbildung 7). Der Geometrie–Manager kontrolliert die Position und Größe der enthaltenen Widgets und achtet auf Kollisionen. Als zwei mögliche Verfeinerungen sind Dialog– und Viewport–Widget aufgezeichnet: Dialog–Widgets werden benutzt, um kurze, textuelle Angaben vom Benutzer abzufragen (vgl. Abbildung 10), und enthalten Label–Widgets (für die Abfrage), Text–Widgets (für die Eingabe des Benutzers) und Command–Button–Widgets (meist „OK" und „Abbrechen"). Ein Viewport–Widget ist ein großes Fenster (eventuell größer als der Bildschirm), in dem ein kleineres Fenster eingebettet ist, durch das man einen Ausschnitt der im großen Fenster angezeigten Information sieht (vgl. Preview–Bereich in Abbildung 7). Horizontale und vertikale Scrollbars ermöglichen, den angezeigten Ausschnitt zu verschieben.

## 5.3 Notwendige Änderungen für den Link–Editor

Da GhostView im Quellcode vorliegt und die GNU–Lizenz eine Bearbeitung gestattet, lag es nahe, den Link–Editor auf der Basis von GhostView zu programmieren. Dabei sollte die Bedienung von GhostView im wesentlichen erhalten bleiben und nur um die entsprechenden Aktionen zum Hinzufügen und Löschen von Links für HelpViewer erweitert werden. Die Syntax der Links und ihre verträgliche Positionierung im PostScript–Code wird dem Benutzer hierbei durch Eingabemasken verborgen. Der Hypertextredakteur muß auf diese Weise keinerlei Kenntnisse über die Signatur der Link–Anweisungen besitzen und benötigt — wie eingangs gefordert — nur ein Vertrautsein mit dem Link–Prinzip.

Das grundlegende Konzept bei der Umprogrammierung ist, GhostView möglichst unverändert in seiner Eigenschaft als Previewer einzusetzen und die eigentliche Verwaltungsarbeit mit den Link–Kommentaren in einer eigenen Datenstruktur — einfacher Text — in erweiternden Prozeduren zu erledigen. Dies erfordert zunächst eine Separation von reinem PostScript–Code und Link–Kommentaren, dann eine Link–Verwaltung, die über den Event–Dispatcher an den Preview gekoppelt ist, und zuletzt ein Zusammenführen von PostScript–Code und Links. Die gänzlich separate Haltung der Hypertextlinks ist wegen der Struktur des Navigators nicht möglich, eine zusätzliche separate Haltung wird aus Konsistenzgründen verworfen.

[26] Alternativen bzw. Ergänzungen zu Aw sind etwa Motif (OSF), Open Look (AT&T) oder SunView (Sun Microsystems).

Der resultierende Link–Editor weist also insgesamt eine dreischichtige Architektur auf:

(1) Die GhostScript–Schicht sorgt für die Umsetzung des PostScript–Programmes in bestmögliche Bitmapdarstellung für die eingesetzte Oberfläche.

(2) Die GhostView–Schicht handhabt den Preview, die Ansicht der in Schicht 1 generierten Bitmaprepräsentation des PostScript–Programmes im eingesetzten Fenstersystem *X11*, und wertet unterstützend die eingebetteten Strukturkommentare aus.

(3) Die Link–Editor–Schicht verwaltet — gesteuert aus der Voransicht heraus — in einer separaten Datenstruktur die Link–Kommentare.

Die für die Implementierung der dritten Schicht notwendige Programmierung wird nachfolgend beschrieben.

### 5.3.1 Vorbereitendes Scannen beim Öffnen der Datei

Beim Öffnen einer Datei durchsucht GhostView den Quellcode nach den auszuwertenden Strukturkommentaren. Der entsprechende Modul, „ps.c", muß also so umprogrammiert werden, daß auch die HelpViewer–spezifischen „%%Link"–Anweisungen beim Scannen erkannt werden, sofern sie enthalten sind. Sie haben auf die Ausgabe des interpretierten PostScript–Programmes im Preview und für dessen Steuerung keine Auswirkung und können deswegen vom PostScript–Quellcode separiert werden. Dies ermöglicht zum einen die weitgehend uneingeschränkte Verwendung aller von GhostView angebotenen Funktionalität und erleichtert zum anderen die Verwaltung der Links, die auf eine separate, überschaubare Datenstruktur konzentriert bleibt. Sicherzustellen ist bei diesem Vorgehen die Konsistenz der Daten: separierte Links dürfen im zugrundeliegenden Code nicht zurückbleiben, da sie sonst nach Veränderungen in der Link–Struktur (etwa nach Löschen) zu Phantomeffekten in HelpViewer führen könnten; sie müssen aber aus der aktuellen Code–Datei gewonnen werden und sind temporär nur während der Bearbeitung zu halten, da Änderungen manuell außerhalb des Link–Editors (eventuell manuell in PageView) sonst unberücksichtigt blieben. In der separaten Link–Datenhaltung ist die Seitenstruktur des PostScript–Dokumentes einzubringen, da die Koordinaten der Links immer auf eine ausgezeichnete Startseite bezogen sind und sich bei Verschieben auf eine andere Seite der Sinn des Links verändert.

Der erweiterte Scanner legt unter Berücksichtigung der genannten Aspekte beim Öffnen der selektierten PostScript–Datei eine Textdatei mit dem gleichen Stammnamen an wie die geöffnete Datei, jedoch durch die Extension „.link" erweitert. Dies ermöglicht, auch nach Systemabbrüchen ggfs. zurückgebliebene Dateien zu identifizieren. Die Implementierung geht im Interesse einer sicheren Konsistenz von einer sorgfältigen Plazierung der bearbeiteten Dateien in der Verzeichnishierarchie durch den Hypertextredakteur aus und überschreibt deshalb ohne Warnung eine ggfs. vorhandene „.link"–Datei mit gleicher Bezeichnung.

Die Struktur der „.link"–Datei ist einfach und orientiert sich weitgehend an der Signatur der Link–Kommentare:

❑ Jeder Link steht in einer eigenen Zeile,

- ❏ erster Bestandteil eines Datensatzes ist das ‹label› der beherbergenden Seite, gewonnen aus dem „%%Page"–Kommentar,
- ❏ zweiter Bestandteil ist die absolute Seitennummer der beherbergenden Seite und
- ❏ die weiteren Bestandteile sind genau die der Signatur der Links entsprechenden, nämlich das spätere Kommentartag „%%Link", der in seiner Funktion ungeklärte, konstante Parameter „v1", der Parameter zur Verhaltenssteuerung der Link–Aktivierung („0" oder „1"), die Koordinaten des begrenzenden Rechtecks, die ebenfalls konstante Angabe „view", ggfs. Pfad und/oder Dateibezeichnung für das Ziel eines Interlinks und die absolute Seitennummer des Ziels.

Als Separator der einzelnen Bestandteile dient konform zur Link–Syntax das Kolon „:". Eine temporäre „.link"–Datei könnte also beispielsweise so aussehen, wie in Abbildung 9 abgedruckt.

```
::1:%%Link:v1:0:91 417 349 437:view::2
::1:%%Link:v1:0:91 387 291 407:view::4
::1:%%Link:v1:0:91 357 317 377:view::6
::1:%%Link:v1:0:92 327 354 347:view::8
::1:%%Link:v1:0:69 3 164 58:view:handbooks/top.toc.handbook:1
::1:%%Link:v1:0:253 3 327 58:view:handbooks/question.handbook:1
::2:%%Link:v1:0:50 1 149 56:view::1
::2:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::3:%%Link:v1:0:50 1 149 56:view::1
::3:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::4:%%Link:v1:0:50 1 149 56:view::1
::4:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::5:%%Link:v1:0:50 1 149 56:view::1
::5:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::6:%%Link:v1:0:50 1 149 56:view::1
::6:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::7:%%Link:v1:0:50 1 149 56:view::1
::7:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::8:%%Link:v1:0:50 1 149 56:view::1
::8:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::9:%%Link:v1:0:50 1 149 56:view::1
::9:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
::10:%%Link:v1:0:50 1 149 56:view::1
::10:%%Link:v1:0:247 1 321 56:view:handbooks/question.handbook:1
```

*Abb. 9: Eine typische, temporäre „.link"–Datei*

Zur Sicherung der Konsistenz der Daten wird in der Implementierung ein Zustandsflag geführt. Es ist bei geöffneter, gefilterter PostScript–Datei gesetzt, ohne geöffnete Dateien nicht (z. B. im Ausgangszustand des Link–Editors direkt nach seinem parameterfreien Aufruf). Soll bei gesetztem Flag eine neue Datei geöffnet werden, oder soll auf die aktuelle Datei ein „Reopen" ausgeführt werden, muß zunächst die im Absatz 5.3.3 beschriebene Misch–Prozedur aufgerufen werden.

### 5.3.2 Einarbeitung und Löschung von Hypertext–Links im Dialog

Die Festlegung neuer Links und das Löschen nicht mehr benötigter, vorhandener Links wird als Dialog über die Maus gesteuert.

Die für einen hinzuzufügenden Link erforderlichen Informationen sind

- ❏ die Seite, auf der der Link verankert ist,
- ❏ die Koordinaten des begrenzenden Rechtecks,
- ❏ das Ziel des Links, zusammengesetzt aus dem optionalen Bezeichner eines Zieldokuments und der notwendigen Angabe einer Zielseite.

Hierbei können die ersten beiden Informationen direkt aus dem Preview gewonnen werden, sie sind in Zustandsvariablen des Viewers zugreifbar, die dritte Angabe muß vom Hypertextredakteur oder der erfassenden Person textuell über ein Dialogformular eingegeben werden.

Das Ausfügen vorhandener Links erfordert die Identifikation des zugehörigen Datensatzes in der „.link"–Datei nach einem geeigneten Schlüssel. Als Schlüssel wird die Verankerungsseite und eine Seiten–Koordinate dienen, die auf Inzidenz in einem der Verankerungsseite zugewiesenen Aktivierungs–Rechteck hin getestet wird. Die Implementierung geht bei geschachtelten Links davon aus, daß der textuell oberste zunächst gelöscht werden soll, beschränkt sich bei der Inzidenzprüfung also darauf, den ersten Datensatz zu finden und nicht alle. Alle benötigten Informationen für das Ausfügen können direkt aus dem Preview gewonnen werden und erfordern keine Eingabe durch den Benutzer.

Der Dialog zum Löschen oder Hinzufügen eines Links kann grundsätzlich als Aktion angesehen werden oder an ein Event im Preview gekoppelt sein. Die Implementierung als Aktion würde einen weiteren Menü–Button an der linken Seite des Applikationsfensters (wie „File", „Page", ...) erfordern, dessen Aktionen dann im Modul „actions.c" zu programmieren wären. Die vorliegende Implementierung wählt die zweite Möglichkeit und koppelt den Dialog an das Ereignis „beliebiger Mausknopf im Preview gedrückt", wofür dann die Programmierung entsprechend im Modul „callbacks.c" durchzuführen ist.

Die Umprogrammierung spezialisiert den ursprünglich reinen Viewer „GhostView" zu einem kombinierten Viewer und Editor, wobei das Editieren vor dem Betrachten im Vordergrund der Arbeit steht. Es bietet sich daher an, auf den ursprünglich angebotenen Viewer–Dienst „Zoom" zu verzichten, der in einem separaten Fenster vergrößerte Ausschnitte aus der aktuellen Seite darstellen konnte, und an seiner Stelle die Editor–Funktion zu implementieren. Der Zoom–Dienst war ebenfalls als Callback–Prozedur realisiert und an das Ereignis des Mausklicks im Preview–Widget gekoppelt, weswegen er in der vorliegenden Implementierung nicht weiter angeboten werden kann.

Die Position der Maus im Viewer–Bereich des Anwendungsfensters wird automatisch verfolgt und liegt in Einheiten des zugrundeliegenden PostScript–Koordinatensystems vor. Der Event–Dispatcher des Systems erkennt außerdem, wenn eine Maustaste im Viewer–Bereich betätigt wird. Tritt dieses Ereignis ein, wird über eine Callback–Prozedur eine Dialogbox auf dem Bildschirm erzeugt, die dem Bediener die Dienste „Hinzufügen eines Links" und „Löschen eines Links" und das „Abbrechen" des begonnenen Vorgangs anbie-

tet. Der Callbackprozedur wird bereits die aktuelle Seite und die Maus–Koordinaten zum Zeitpunkt des Events „Klick" übergeben.

Je nach dem interaktiv ausgewählten Dienst wird der Benutzer nun entweder zum Anklicken der zweiten Eckkoordinate für den Link–Bereich im Preview aufgefordert oder die Inzidenzprüfung in der Liste der vorhandenen Links wird gestartet. Die Inzidenzprüfung evaluiert dabei mit den im Event ermittelten Werten für aktuelle PostScript–Seite und x– und y–Koordinate der Maus den booleschen Ausdruck

$$\exists Zeile_{.link} : (x_l \le x_m \le x_r) \wedge (y_l \le y_m \le y_r) \wedge (Seite_{PostScript} = Seite_{.link})$$

wobei $x_l, x_r, y_l, y_r$ Koordinaten aus den Zeilen der „.link"–Datei sind, $x_m$ und $y_m$ die übergebenen Koordinaten des Maus–Events und $Seite_{PostScript}$ und $Seite_{.link}$ die entsprechenden Seitenschlüssel.

Die Steuerung der Dienste geschieht intern wieder durch ein Flag, das den Zustand des Editor–Systems modelliert. Es wird beim ersten Klick im Preview–Bereich gesetzt. Entscheidet sich der Bediener für das Löschen eines Link–Eintrags wird es nach dem Löschvorgang zurückgesetzt, ganz gleich ob dieser erfolgreich oder erfolglos verlaufen ist. Der positive und negative Erfolg eines Löschvorgangs wird in einem Meldungsfenster (einem Label–Widget der Athena–Hierarchie) angezeigt.

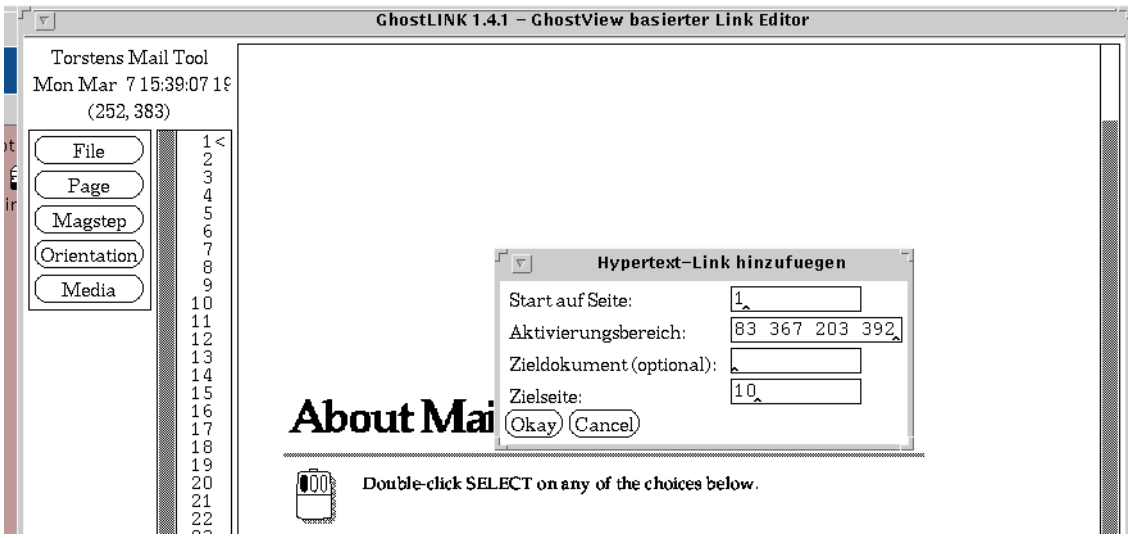


*Abb. 10: Die Dialogmaske zum Hinzufügen eines Hypertext–Links*

Wählt der Bediener das Hinzufügen eines Links, bewirkt das zuvor gesetzte Flag, daß die zuerst angeklickten Koordinaten durch einen nun abgefangenen, zweiten Klick nicht überschrieben werden, sondern in ein Variablenpaar für die zweite Begrenzungsecke des einzufügenden Links gelegt werden. Die Eck–Koordinaten werden automatisch so sortiert, daß sie der vom Link–Kommentar erwarteten Reihenfolge genügen, auch wenn sie in anderer Form eingeklickt wurden, etwa zunächst „rechts–unten" und danach „links–oben". Nun wird ein Dialogformular (Dialog–Widget der Athena–Hierarchie) aufgerufen, das die ausgewählten Koordinaten in der Link–Form ausgibt und über Text–Ein-

gabefelder vom Bediener im Klartext die Angabe der eventuellen Zieldatei und in jedem Fall der Zielseite für den Link anfordert; Abbildung 10 zeigt es. Die Angaben des Link–Ursprungs werden vom System eingetragen, das Dialogfeld für die Koordinaten des Selektionsbereichs ist jedoch editierbar, so daß Justierungen durch den Bediener textuell möglich sind. Die Eingabefelder passen sich einer getippten Eingabe in ihrer Ausdehnung an, so daß auch die Angabe eines langen Pfades im dritten Dialogfeld ohne Schwierigkeiten möglich ist. Das Dialogformular verfügt über die Buttons „Okay" und „Cancel". Bei Auswahl des „Okay"–Knopfes wird das Vorhandensein der Zielseite überprüft, der entsprechende Link–Text in der „.link"–Datei generiert und danach das Statusflag zurückgesetzt. Die neu generierte Link–Zeile wird dabei jeweils unten an die Textdatei angefügt. Der „Cancel"–Button bewirkt ein sofortiges Zurücksetzen des Statusflags, wobei keine Änderungen vorgenommen werden.

Wird während des gesetzten Status–Flags „Hinzufügen begonnen" eine Aktion ausgeführt, die die angezeigte Seite verändert (Blättern, Reopen, ...), wird das Flag sofort zurückgesetzt und der Hinzufüge–Vorgang abgebrochen.

### 5.3.3 Einbringen der Link–Kommentare in den PostScript–Code

Beim Beenden des Hypertext–Editors und beim Öffnen einer neuen Datei wird je nach dem Zustand des in 5.3.1 beschriebenen Bearbeitungsflags die Einarbeitung der separierten Links aus der „.link"–Datei in den bereinigten PostScript–Quellcode ausgeführt.

Dazu wird zunächst mit einem UNIX–Systemaufruf die Textdatei mit den Link–Einträgen sortiert. Als Schlüssel der Sortierung werden dabei ausschließlich die ersten 15 Zeichen jeder Zeile verwendet. Dadurch wird verhindert, daß Koordinatenwerte in die Sortierung eingehen, selbst bei Fehlen jeglicher Seitenangabe in einer Link–Zeile. Da das systemeigene Sortieren ein stabiles Verfahren ist, ist sichergestellt, daß die Reihenfolge der Links einer Seite in der natürlichen, chronologischen Folge erhalten bleibt, die Links einer Seite dennoch zusammengeführt werden und die Seiten in aufsteigender Folge auftreten. Unter dieser Voraussetzung geschieht die Mischung durch Vergleich der Seitenkomponenten aus der „.link"–Datei mit dem „%%Page"–Kommentar im PostScript–Quellcode bei *einem* sequentiellen Durchlauf durch die einzelnen Zeilen des PostScript–Programms. Wird eine Übereinstimmung festgestellt werden alle Link–Zeilen für die betreffende Seite bereinigt und in der für HelpViewer notwendigen Form nach dem „%%Page"–Kommentar im Code kompakt eingefügt. Diese Positionierung beschränkt die Funktion des HelpViewer nicht, sie erschwert allenfalls dem menschlichen Leser des „angereicherten" Programms die Zuordnung von Link zum eventuell als Auslöser dienenden Text.

# 6 Redaktionelle Arbeit und Navigation im Einsatz des Hypertext–Systems

An den Schluß dieser Arbeit sollen noch kurze Überlegungen für die Arbeit des Hypertextredakteurs gestellt werden, die aus der Erfahrung bei den eigenen Versuchen resultieren. Die kurze Betrachtung fiktiver Benutzungsverhalten des Hypertextsystems durch Konstrukteure, Redakteure, Techniker und mittelbar beteiligte Personen schließt sich an.

## 6.1 Architektur des informellen Materials

Das Schrifttum, das für die Vernetzung in Hypertext vorgesehen ist, ist überwiegend in mehrseitige Dokumente gegliedert. Es ist naheliegend, diese Dokumentstruktur beizubehalten. Da jedoch der für die Navigation eingesetzte HelpViewer beim Aufruf die Angabe einer aufzuschlagenden Seite nicht erlaubt, kann es notwendig sein, ein Dokument auf mehrere PostScript–Dateien zu verteilen. Dies erfordert insbesondere eine vorhergehende genaue Bestimmung der Zielseiten für die einzelnen Prozeßschritte.

Soll ein Prozeßschritt den Einstieg in mehrere Dokumente ermöglichen, kann es sinnvoll sein, ein eigenständiges Inhalts–Dokument für diesen Prozeßschritt zu erstellen, in das aus dem Slot heraus eingesprungen wird, und aus dem heraus auf die einzelnen Seiten der tatsächlichen Dokumente gesprungen wird. Da in einer solchen Realisierung bei der zweiten Verzweigung bereits der volle Mechanismus der Interlinks greift, kann auch die Fragmentierung von Dokumenten auf diese Weise vermieden werden.

Die Anlage und Fortschreibung eines eventuell hierarchischen, mehrstufigen Dokumentverzeichnisses, das ebenfalls in das Hypertext–Netz integriert ist, ermöglicht dem Leser, auch bei direktem Einstieg in die Navigation über die UNIX–Shell (nicht aus der Prozeßmodellierung heraus) eine Übersicht zu gewinnen, und hält die Zahl der Einstiegspunkte in das Netz überschaubar, ohne jedoch die Möglichkeit zu verschließen, in ein ausgewähltes Dokument direkt einzusteigen.

Einen raschen Überblick über ein bereits bearbeitetes Dokument und die in ihm enthaltenen Links kann sich der Hypertextredakteur mit Hilfe der folgenden UNIX–Pipe auf die betroffene Datei verschaffen:

```
cat ‹PostScript-Datei›|egrep ´^%%(Page|Link):´|more
```

beziehungsweise, bei Interesse an allen Strukturkommentaren:

```
cat ‹PostScript-Datei›|egrep ´^(%%|%!)´|more
```

## 6.2 Generierung von PostScript–Dateien auf den gängigen Textverarbeitungssystemen

Ein Teil der zu vernetzenden Dokumente wird sicherlich auf dem Industriestandard–PC erstellt sein. Ein dort weit verbreitetes Textverarbeitungssystem ist „Word for Windows" der Microsoft Corporation. PostScript–Dateien aus PC–Dokumenten, insbesondere aus den Windows–Applikationen lassen sich einfach erstellen. Es ist nur ein Treiber für einen PostScript–Drucker anzuwählen und die Ausgabe statt auf den Druckeranschluß in eine Datei umzuleiten. Bei den Microsoft–eigenen Druckertreibern läßt sich als Option für den PostScript–Druck zwar „gemäß der Adobe Dokumentstrukturierung" anwählen, leider wird jedoch entgegen der Angabe im identifizierenden Kommentar der Ausgabedateien keine saubere Dokumentstruktur erzeugt: die verzichtbaren Header–Kommentare des Prologues sind komplett vorhanden, jedoch die zur Konformität notwendige Trennung und Numerierung der einzelnen Dokumentseiten über den „%%Page"–Kommentar wird unterlassen. Am Ende einer jeden Seite wird allerdings explizit der showpage–Operator oder eine showpage enthaltende Page–Trailer–Prozedur eingefügt. Für den Hypertextredakteur, der viele PC–Dokumente zu verarbeiten hat, ist es daher hilfreich ein kleines UNIX–Script zu erstellen, das das PC–PostScript–Programm durchscannt und dabei jedes showpage durch die Sequenz „showpage – ‚Newline' – %%Page: ‹n›" mit fortlaufendem ‹n› ersetzt. Das angesprochene Problem hat seine Wurzel im Microsoft–PostScript–Druckertreiber, kann also meist nicht durch Einsatz eines anderen als des genannten Textverarbeitungssystems vermieden werden.

Das genannte Textsystem ist in gleicher Form unter der Oberfläche „Presentation Manager" unter „OS/2" und unter „System 7" auf dem Apple MacIntosh erhältlich. Die unter diesen Betriebssystemen erhältlichen PostScript–Druckertreiber konnten nicht getestet werden.

Eine weitere im Kernforschungszentrum Karlsruhe weit verbreitete Architektur, die als Quelle für Hypertextdokumente dient, ist die der XEROX–Arbeitsstationen unter der grafischen Benutzeroberfläche „ViewPoint" oder „GlobalView". Bislang standen für dieses betagte System keine PostScript–Treiber zur Verfügung, da XEROX eine eigene Seitenbeschreibungssprache mit der Bezeichnung „Interpress" entwickelt hat und für die Ansteuerung der proprietären Drucker einsetzt, die jedoch im Gegensatz zu PostScript auf Bitmap–Basis arbeitet und sich deshalb am Markt nicht durchsetzen konnte. Die „GlobalView"–Umgebung wurde aber mittlerweile auf UNIX und X11 implementiert, so daß auch PostScript–Treiber verfügbar werden. Erfahrungen mit GlobalView–Dokumenten und ihrer Ausgabe in Adobe–konforme Dateien konnten jedoch bisher nicht gesammelt werden.

Eine dritte, weit verbreitete Dokumentquelle ist das wissenschaftlich orientierte in der „Public Domain" (kostenlos oder gegen Ersatz der Materialkosten für die Auslieferung) erhältliche „TEX", das auf verschiedensten Plattformen (Workstation, Großrechner, ISA–PC, MacIntosh, Atari, ...) zum Einsatz kommt. TEX erfaßt Texte mit eingebetteten For-

matierungskommandos, die in einem Interpretationslauf in eine sogenannte DVI[27]–Datei umgesetzt werden. In einem weiteren Schritt wird dann eine Druckdatei für einen spezifischen Drucker erzeugt. Da TEX nach seiner Einführung zunächst überwiegend auf UNIX–Workstations zum Einsatz kam, ist ein PostScript–Treiber im Standardumfang enthalten. Mit dem Programm „dvips" können vollkonforme Quelldateien ohne Schwierigkeiten generiert werden.

## 6.3 Dateiplazierung und Benennung

Die Sun–Workstations, von denen aus das Hypertext–Netz genutzt wird, halten sämtliche Plattendaten in einem NFS[28]–System.

Es ist daher empfehlenswert, alle Hypertext–Dokumente in ein öffentlich zugängliches Verzeichnis zu stellen, das die interessierten Benutzer in den Pfad ihrer Arbeitsumgebung aufnehmen können. Die Dateien sollten für die Benutzer „read only" sein, also etwa mit den UNIX–Rechten „-rw-r--r--" (Schreib– und Leserecht für den Eigentümer, Leserecht für die Gruppe des Eigentümers und alle anderen Benutzer) versehen sein.

Ist hinreichend freie Plattenkapazität vorhanden, können zusätzliche Kopien der Dokumente in einem speziellen, ausschließlich den Redakteuren zugänglichen Verzeichnis gehalten werden. Sind an einem Dokument Änderungen notwendig, werden sie vom Hypertextredakteur in den kopierten Dokumenten durchgeführt und erst nach Korrektur in das öffentliche Verzeichnis eingestellt. Auf diese Weise wird der Navigationsbetrieb im geringst möglichen Umfang behindert.

Störungen des Navigationsbetriebes beim Hinzufügen neuer Dokumente in das Netz können nicht auftreten, wenn die hinzugefügten Dokumente neben Intralinks nur Ausgangspunkte von Interlinks enthalten und nicht selbst als Ziel von Interlinks aus bereits bestehenden Dokumenten eingefügt werden müssen. Müssen solche Verweise in bestehende Dokumente aufgenommen werden, wird der Navigationsbetrieb nur dann gestört, wenn ein Viewprozeß gerade in einem der zu verändernden Dokumente positioniert ist.

Zur Unterscheidung reiner PostScript–Dateien von den überarbeiteten Hypertext–Dateien verwendet Sun Microsystems Inc. verschiedene Datei–Suffixe, die selbstverständlich nicht zwingend vorgeschrieben sind: die pure PostScript–Datei führt als letzten durch Punkt abgetrennten Teilbezeichner „ps", das redaktionell überarbeitete Dokument an dieser Stelle „handbook". Die Befolgung dieser Konvention hat sich als hilfreich erwiesen.

---

[27] DVI steht für „device independent", zu deutsch „geräteunabhängig"

[28] NFS steht für „Network File System", ein Konzept von Sun Microsystems Inc., das die Platten verschiedener, vernetzter Workstations in ein großes Dateisystem integriert. Für den Benutzer existiert nur eine Katalog–Hierarchie, die von jeder Workstation aus gleich erscheint. Die tatsächliche Plazierung einer Datei auf einer speziellen Platte bleibt verdeckt.

## 6.4 Ein mögliches Nutzungsszenario von Prozeßmodellierung und Hypertextkomponente

Wie bereits in der Motivation der Aufgabe beschrieben, sind verschiedene Gruppen von Benutzern zu unterscheiden. Ein Benutzer gehört dabei jedoch nicht zwangsläufig nur einer Gruppe an, er tritt vielmehr zu verschiedenen Zeitpunkten in verschiedenen Rollen auf. Die Rollen, die in diesem abschließenden Szenario unterschieden werden sollen, sind:

(a) Die des ***Konstrukteurs*** einer Mikrostruktur, der sowohl die Prozeßmodellierung als auch die Hypertextkomponente als Werkzeug für seinen kreativen Entwurfsprozeß einsetzt.

Für ihn kommt die Nutzung der Hypertextkomponente in dreierlei Weise in Betracht: zunächst ggfs. als simple On–Line–Anleitung für die Bedienung des Prozeßmodells. Hier werden ihm z. B. Hinweise gegeben, wie die Prozeßmodellierung gestartet werden kann, wie mit Hilfe der Maus oder anderer Interaktionsmechanismen innerhalb der Prozeßschritte navigiert werden kann, wie bestimmte Informationen am Bildschirm angezeigt oder wie Prozeßparameter verändert werden können. Diese Information wird sinnvoll in einem vom Terminal aufgerufenen, selbständigen View–Prozeß angeboten. Für seine eigentliche Entwurfstätigkeit kann der Konstrukteur in einem oder zwei weiteren selbständigen View–Prozessen über geeignete Artikulationspunkte des Hypertextnetzes insbesondere auf die Dokumente der früher eingeführten Klassen (1) und (4) zugreifen. Daneben wird er beim Einsatz des Prozeßmodells aus der Ansicht bestimmter Teilprozeßknoten zusätzliche Information, etwa der Klasse (3), betrachten wollen, die ihm die gekoppelte Hypertextnavigation in einem weiteren eigenständigen Prozeß liefert.

(b) Eine weitere Rolle ist die des ***Hypertextautors und Hypertextredakteurs,*** die jeder Konstrukteur und viele Techniker zeitweise einnehmen können. Zur koordinierten Erweiterung des Hypertext–Netzes und zur Vermeidung von Wildwuchs und „Hypertext–Leichen", also unerreichbaren Inselknoten oder Sackgassen, wird eine organisatorische Maßnahme einen eingeschränkten Personenkreis für diese Rolle bestimmen.

Für diesen Personenkreis kommt die Nutzung des vorhandenen Hypertext–Netzes in möglicherweise mehreren, selbständigen View–Prozessen in Betracht, in denen unterschiedliche Dokumente auf ihre Eignung als Start oder Ziel der hinzuzufügenden Links untersucht werden. Daneben wird für die neueinzugliedernden Dokumente ein oder – recht unwahrscheinlich – sogar mehrere Link–Editor–Prozesse ablaufen und in einem Lauf der LIGA–Prozeßmodellierung werden neue Aufrufe in den entsprechenden Slots eingetragen.

(c) Die dritte Rolle ist der ***Techniker in der Mikrostrukturfertigung,*** der zur Nachverfolgung seiner Tätigkeit im Gesamtablauf eventuell einen Prozeßmodellierungs–Prozeß gestartet hat, aus dem er Informationen zu den durchzuführenden Prozeßschritten erhält.

Die Nutzung des Hypertextnetzes kann bei ihm einerseits wieder auf der Ebene der On–Line–Hilfe zur Bedienung des Prozeßmodells geschehen, andererseits wird er die für die einzelnen Prozeßschritte verfügbaren Zusatzinformationen der Klasse (2) aus der Modellierung heraus aufrufen.

(d) Die vierte Rolle betrachtet den neuen ***Mitarbeiter, der sich einen Überblick zu verschaffen sucht,*** oder eine Person, die für zusammenfassende Darstellungen gezielte Informationen sammelt.

Die Nutzung des Hypertext–Systems geschieht hier voraussichtlich unabhängig vom Prozeßmodell in einem oder mehreren, selbständigen View–Prozessen von verschiedenen Artikulationspunkten her besonders auf Dokumenten der Klassen (4) und (2).

## 6.5 Zusammenfassung und Ausblick

In der vorliegenden Studienarbeit wurde ein Hypertext–Konzept vorgestellt, das über zwei einfach zu bedienende Werkzeuge — den Navigator und den Link–Editor — die Möglichkeiten eröffnet, zu verschiedenen Themen ein Hypertext–Netz zu benutzen und zu erstellen. Dabei wurde berücksichtigt, daß das System einerseits selbständig, andererseits auch aus einer interaktiven Anwendung heraus nutzbar sein soll. Bei der Durchführung dieser Arbeit war die betrachtete, interaktive Anwendung die Prozeßmodellierung der LIGA–Mikrostrukturfertigung. Unter dem Aspekt des Neuaufbaus eines solchen Hypertextnetzes mußte insbesondere beachtet werden, daß die möglichen Dokumente auf verschiedenen Architekturen erstellt werden. Das Hypertext–System muß also auf einem portablen, allgemein erzeugbaren Dateiformat — PostScript — arbeiten.

In dieser Arbeit wird ein reines Hypertext–System betrachtet; eine mögliche Erweiterung könnte ein Hypermedia–System sein, das neben der zweidimensionalen Grafik auch akustische Informationen oder animierte Bildszenen aufnehmen kann.

# 7 Anhang

## 7.1 Literatur

/Adobe90a/ Adobe Systems Incorporated. *PostScript Language Reference Manual.* Sixteenth Printing, June 1990. Addison–Wesley, Reading, Massachusetts, 1990.

/Adobe90b/ Adobe Systems Incorporated. *PostScript Language Tutorial and Cookbook.* Sixteenth Printing, July 1990. Addison–Wesley, Reading, Massachusetts, 1990.

/Brauch92/ Ilko Brauch, Horst Eggert, Klaus Peter Scherer, Peter Stiller. *Einsatz wissensbasierter Methoden für Konstruktion, Fertigung und Test von LIGA–Mikrostrukturen.* Kernforschungszentrum Karlsruhe GmbH, 1992.

/Gloor90/ Peter A. Gloor, Norbert A. Streitz (Hrsg.). *Hypertext und Hypermedia.* Informatik–Fachberichte 249. Springer–Verlag, Berlin, 1990.

/Jones91/ Oliver Jones. *Einführung in das X–Window–System.* Aus dem Amerikanischen übersetzt. Carl Hanser Verlag, München, 1991.

/Maurer91/ Hermann Maurer (Hrsg.). *Hypertext/Hypermedia '91.* Informatik–Fachberichte 276. Springer–Verlag, Berlin, 1991.

/Nye90/ Adrian Nye, Tim O'Reilly. *X Toolkit Intrinsics Programming Manual.* The X Window System. Volume Four. Second Edition for X11, Release 4. O'Reilly & Associates, Inc., Sebastopol, California, 1990.

/Schneider91/ Hans–Jochen Schneider (Hrsg.). *Lexikon der Informatik und Datenverarbeitung.* 3., aktualisierte und wesentlich erweiterte Auflage, 1991. R. Oldenbourg Verlag, München, 1991.

## 7.2 Die “GNU General Public License”

**GNU GENERAL PUBLIC LICENSE**
**Version 2, June 1991**



**Preamble**

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

(1) copyright the software, and

(2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

**GNU GENERAL PUBLIC LICENSE**
**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means

either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty–free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

**NO WARRANTY**

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

**Appendix: How to Apply These Terms to Your New Programs**

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the
Free Software Foundation; either version 2 of the License, or (at your
option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANT-
ABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it under
certain conditions; type ‘show c’ for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items — whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989

Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

## 7.3 Auszug aus der Sprachbeschreibung von GhostScript

This file is part of Ghostscript.

Ghostscript is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY. No author or distributor accepts responsibility to anyone for the consequences of using it or for whether it serves any particular purpose or works at all, unless he says so in writing. Refer to the Ghostscript General Public License for full details.

Everyone is granted permission to copy, modify and redistribute Ghostscript, but only under the conditions described in the Ghostscript General Public License. A copy of this license is supposed to have been given to you along with Ghostscript so you can know your rights and responsibilities. It should be in a file named COPYING. Among other things, the copyright notice and this notice must be preserved on all copies.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

This file, language.doc, describes the Ghostscript language.

For an overview of Ghostscript and a list of the documentation files, see README.

The Ghostscript interpreter, except as noted below, is intended to execute properly any source program written in a language defined by reference to the December 1990 printing of the PostScript Language Reference Manual (Second Edition) published by Addison-Wesley (ISBN 0-201-18127-4). The Ghostscript language includes the following elements of the PostScript (TM) language:

- The full PostScript Level 1 language, as also defined in the first edition of the PostScript Language Reference Manual, ISBN 0-201-10174-2, Addison-Wesley, 1985.
- The CMYK color, file system, version 25.0 language, and miscellaneous additions listed in sections A.1.4, A.1.6, A.1.7, and A.1.8 of the Second Edition respectively.
- The Display PostScript extensions listed in section A.1.3 of the Second Edition, but excluding the operators listed in section A.1.2, and also excluding setbbox, xshow, xyshow, and yshow. These extensions are only available if the dps feature was selected at the time that Ghostscript was compiled and linked.
- A few other PostScript Level 2 operators, listed below.

Ghostscript also includes a number of operators defined below that are not in the PostScript language.

Stub facilities

The following operators, while provided in the current release, have only a partial or dummy implementation.

Character and font operators:

```
cshow, rootfont, setcachedevice2
```

Graphics state operators:

```
currentblackgeneration, currentcmykcolor, currentcolorscreen, cur-
rentcolortransfer, currenthalftonephase, currentundercolorremoval, set-
blackgeneration, setcmykcolor, setcolorscreen, setcolortransfer,
sethalftonephase, setundercolorremoval, currenthalftone, sethalftone,
setbbox
```

Interpreter parameter operators:

```
setucacheparams, ucachestatus
```

Path construction operators:

```
ucache
```

Virtual memory operators:

```
currentshared, scheck, setshared, setvmthreshold, shareddict, Shared-
FontDirectory, vmreclaim
```

Level 2 operators

The following PostScript Level 2 operators are available in Ghostscript. Unless otherwise noted, these are only available if the level2 feature was selected at the time that Ghostscript was compiled and linked.

File operators:

`filter`
(not all filters are implemented, and most of the implemented ones are only available if the filter feature was selected when Ghostscript was built)

Graphics state operators:

```
currentcolor, currentcolorspace, setcolor, setcolorspace (for DeviceGray,
DeviceRGB, and DeviceCMYK only) currentstrokeadjust, setstrokeadjust
(even if level2 is not selected)
```

In addition, Ghostscript supports the following Level 2 facilities:

- Use of a string with the status operator (even if level2 is not selected);
- Use of a dictionary with the image and imagemask operators;
- Use of a string or a file as data source with the image, imagemask, and colorimage operators (even if level2 is not selected).

**Ghostscript–specific additions**

[...]

## 7.4 Von GhostView ausgewertete Strukturkommentare und verarbeitete Formate

### 7.4.1 Ausgewertete Strukturkommentare

%!PS-Adobe-<real> [EPSF-<real>]
%%BoundingBox: <int> <int> <int> <int>|(atend)
%%CreationDate: <textline>
%%Orientation: Portrait|Landscape|(atend)
%%Pages: <uint>|(atend)
%%PageOrder: Ascend|Descend|Special|(atend)
%%Title: <textline>
%%DocumentMedia: <text> <real> <real> <real> <text> <text>
%%DocumentPageSizes: <text>
%%EndComments

%%BeginPreview
%%EndPreview

%%BeginDefaults
%%PageBoundingBox: <int> <int> <int> <int>|(atend)
%%PageOrientation: Portrait|Landscape
%%PageMedia: <text>
%%EndDefaults

%%BeginProlog
%%EndProlog

%%BeginSetup
%%PageBoundingBox: <int> <int> <int> <int>|(atend)
%%PageOrientation: Portrait|Landscape
%%PaperSize: <text>
%%EndSetup

%%Page: <text> <uint>
%%PageBoundingBox: <int> <int> <int> <int>|(atend)
%%PageOrientation: Portrait|Landscape
%%PageMedia: <text>
%%PaperSize: <text>

%%Trailer
%%EOF

%%BeginDocument: <text> [<real>[<text>]]
%%EndDocument

%%BeginBinary: <uint>
%%EndBinary

%%BeginData: <uint> [Hex|Binary|ASCII[Bytes|Lines]]
%%EndData

### 7.4.2 Formatbezeichnungen und Papiergröße in „Punkt“

| | |
|---|---|
| Letter | 612x792 |
| LetterSmall | 612x792 |
| Tabloid | 792x1224 |
| Ledger | 1224x792 |
| Legal | 612x1008 |
| Statement | 396x612 |
| Executive | 540x720 |
| A3 | 842x1190 |
| A4 | 595x842 |
| A4Small | 595x842 |
| A5 | 420x595 |
| B4 | 729x1032 |
| B5 | 516x729 |
| Folio | 612x936 |
| Quarto | 610x780 |
| 10x14 | 720x1008 |