Foraus

Projektbericht

Projektleitung:
Prof. Bernd Krieg-Brückner

Projektbetreuung:
Michael Fröhlich, Mattias Werner

Projektteilnehmer:
Achim Mahnke, Andreas Hinken, Andree Hähnel, Bernd Rattey,
Christian Schaefer, Frank Hettling, Guido Frick, Gunnar Kavemann,
Jan Hiller, Jens Warnken, Jens-Martin Brinkmann, Kai Hofmann,
Michael Jäschke, Rainer Schmidtke, Tarik Ali, Trinh Le,
Van Son Le, Volker Schmidtke

Universität Bremen

Inhaltsverzeichnis

1. FORAUS - wie alles anfing

Michael Fröhlich

1. April 1992, Spätnachmittag, IC 520 "Spessart", irgendwo zwischen Bonn und Bremen[1]. Wir, das sind Prof. Dr. Bernd Krieg-Brückner[2], Mattias Werner, zwei weitere Kollegen unserer Arbeitsgruppe und ich, befinden uns auf der Rückfahrt vom Workshop des KORSO-Projektes. Über drei Tage lang diskutierten wir in einer Ansiedelung namens Meckenheim bei Bonn mit Forschungspartnern aus der ganzen Republik über das weitere Vorgehen im Projekt. Für Mattias und mich ist es nach einem halben Jahr Anstellung in der Gruppe die erste größere Dienstreise dieser Art. Abgespannt und müde freut man sich auf Zuhause; die stundenlangen Marathonsitzungen und abendlichen Trinkgelage fordern ihren Tribut.
Nach einer umfangreichen Reflexion und geistigen Nachbearbeitung der gehörten Vorträge schlägt das Thema der Diskussion unserer Reisegruppe urplötzlich um. BKB berichtet von unglaublichen Geschehnissen in der heimatlichen Universität. Für die alljährlich beginnenden studentischen Projekte, die von jeder Studentin und von jedem Studenten gewählt werden und die sich über das gesamte Hauptstudium erstrecken, liegen bisher lediglich zwei Nennungen vor, welche beide nicht in der praktischen Kerninformatik angesiedelt sind. Mattias und mir stockt der Atem. Waren wir es nicht, die bis vor kurzem selber noch Studenten an dieser Uni gewesen sind und von der Idee begeistert waren, in einem Team an einem wirklichkeitsnahen Software-Problem zu arbeiten, anstatt im heimischen Kämmerlein an Miniaturalgorithmen zur Befriedigung von Übungszettel-Revisoren zu werkeln?
Schnell beginnen wir über Aktionsprogramme zur Rettung der praktischen Informatik in Bremen nachzudenken. In einem Kraftakt soll unsere Arbeitsgruppe gleich zwei Projekte anbieten, eines davon im klassischen Gebiet unserer Arbeitsgruppe angesiedelt, dem Übersetzerbau. Mit dem anderen Projekt soll hingegen Neuland betreten werden. Wir überlegen, wie der Schwerpunktbereich von Mattias und mir, die grafische Darstellung von Strukturen, hierbei sinnvoll eingebracht werden kann. Viele Vorschläge werden gemacht und wieder verworfen, bis die Überlegung aufkommt, ein fortschrittliches Textverarbeitungssystem zu realisieren. Die Struktur eines Dokumentes mit seinen Kapiteln, Abschnitten, Absätzen aber auch Querverweisen und Verzeichnissen scheint sich in hervorragender Weise für eine Visualisierung zu eignen. Wir phantasieren, wie man beispielsweise die Reihenfolge ganzer Textpassagen auf simpelste Weise mit der Maus reorganisieren kann, indem Knoten des visualisierten Dokument-Baumes einfach verschoben werden. Schnell kommen uns weitere Ideen, die wir bei der täglichen Arbeit mit unseren Textverarbeitungssystemen schon immer vermißt haben:

[1] Es handelt sich hier nicht um einen Aprilscherz. Die besagte Bahnfahrt fand tatsächlich an diesem Tag statt. Wird sich dieser Umstand als böses Omen für das Projekt FORAUS herausstellen?

[2] Dieser Name erscheint uns im Gegensatz zur Person unsympathisch, weil länglich. Wir entscheiden uns daher von nun an für die Floskel BKB.

- Eine Strukturorientierung mit vorher definierbaren Dokumenttypen wie Brief, Artikel oder Buch soll komplizierte Formatierungsarbeiten beim Schreiben von Texten überflüssig machen und zu einheitlicheren Dokumenten führen.
- Mehrere gleichzeitig bereitgestellte Sichten auf das Dokument mit einer Struktursicht, mit Sichten in unterschiedlichen Formatierungen oder mit einer TEX-artigen Steuerzeichensicht sollen das Edieren von Texten erleichtern.
- Auch Fragen des Mehrbenutzerbetriebes, bei dem mehrere Autorinnen oder Autoren gleichzeitig an einem Text arbeiten, sind es wert, untersucht zu werden.
- Erprobte Techniken des Übersetzerbaus wie die Attributierung könnten eingesetzt werden, um die Effizienzprobleme bei der Programmierung unserer Ideen in den Griff zu bekommen.

Wieder an der Universität arbeiten wir in der darauf folgenden Zeit mit Hochdruck an der Ausarbeitung unserer Vorstellungen, um am Stichtag 11. Juni 1992 allen Studierenden des Jahrgangs unseren Projektvorschlag präsentieren und zur Wahl stellen zu können. Je mehr Mattias und ich sich mit dem Thema beschäftigen, umso mehr kommt bei uns der Verdacht auf, daß sich dieses Projekt wegen seines starken Praxisbezuges wie ein Magnet auf die Leute auswirken wird. Die drei Konkurrenzprojekte klingen für uns in der Ankündigung recht theorielastig oder spezialisiert. Wir malen uns unerträgliche Arbeitssituationen aus, bei denen ein Großteil der über 100 Studentinnen und Studenten unser Projekt belegt, obwohl vielfach eine Anzahl von nur 20 Personen als sinnvolle und verträgliche obere Grenze angesehen wird. Mattias und ich beschließen daher, unseren Vorstellungsvortrag betont sachlich, ja sogar abschreckend zu gestalten, um der vermuteten Flut von Interessenten Herr zu werden. Eine Idee, die wir noch bereuen werden.
Fest der Überzeugung, den Balanceakt zwischen *"Begeisterung wecken"* und *"abschrecken"* gemeistert zu haben, sehen wir dem Tag der Vorstellung entgegen. Mit Schlagworten wie *"Keine Implementierung in imperativen Programmiersprachen wie C oder Modula-2"* und dem durchrechnen eines gruseligen Attributierungsbeispiels schocken wir das Auditorium. Doch der Schuß geht unerwartet nach hinten los. Wir haben mit unseren Abschreckungsbemühungen maßlos übertrieben, denn bei der anschließenden Abstimmung zeigen sich keine zehn Leute bereit, sich für unser Projekt zu entscheiden. Für uns stürzt eine Welt zusammen, denn mit nur einer Handvoll Teilnehmern lassen sich die ehrgeizigen Ziele kaum verwirklichen. Doch diese Abstimmung hat noch keine endgültige Verteilung zur Folge und Mattias und ich setzen nun auf den Semesterbeginn im kommenden Oktober, wo die wirkliche Zuordnung geschehen wird.
Bis dahin entwerfen wir ein großes Plakat, welches überall gut sichtbar in der Uni aufgehängt wird. Hierauf werben wir noch einmal für das Projekt, aber diesmal auf ganz andere Weise mit all den schönen, aktuellen Schlagworten, die Studierende besonders interessant finden und kündigen dabei unter dem Thema *"die Wahrheit über FORAUS"* eine erneute Einführungsveranstaltung für alle Unschlüssigen und Neugierigen an. Das bleibt nicht ohne Wirkung, denn zum ersten Termin im Semester kommen trotz der frühen Zeit von 8 Uhr tatsächlich über 20 Studentinnen und Studenten. Mattias und ich schöpfen neue Hoffnung und zeigen, wie man auch in einer funktionalen Sprache wie ASpecT systemnah grafische Benutzungsoberflächen programmieren kann, stellen anhand von Folien vor, wie wir uns eine *Textverarbeitung mit unterschiedlichen Sichten* denken und machen zum Abschluß einige Demonstrationen am Rechner.
Nach unserer verpatzten Premiere ist diese neue Vorstellung ein voller Erfolg, denn von den Anwesenden werden, wie sich später herausstellen wird, alle (bis auf wenige Ausnahmen) im Projekt FORAUS bis zum Ende mitarbeiten. Somit können wir unsere Mission zur Rettung der praktischen Informatik fortführen...

2. Projektverlauf

Jens Warnken

In diesem Kapitel soll in chronologischer Reihenfolge der Projektverlauf aufgezeigt werden. Dabei wird nicht nur auf die Ergebnisse der Plenen oder der im Projektverlauf entstandenen Arbeitsgruppen eingegangen, sondern auch die Aktivitäten außerhalb des eigentlichen Projektbetriebes beschrieben. Gerade diese Aktivitäten machen das Projektstudium zu einem unvergeßlichen Erlebnis in der Gruppe.

2.1 1. Semester Okt. '92 – März '93

Es war einmal am 15. Oktober 1992, 8:00 Uhr, MZH 2490. Da begab es sich, daß sich eine Gruppe von Fünfsemesterfrischlingen traf, um ihr Projekt anzutreten. Unsicher durch das, was jetzt wohl kommen mag, beäugten sich die ForauSianerInnen. Die meisten hatte man schon irgendwo gesehen, mit dem einen oder anderen auch gesprochen oder eine Lösung zu einem Algorithmen & Programmierung IV-Aufgabenzettel getauscht. Der unbekannte Rest (wie alle anderen auch) stellte sich dann nach einem von den Projektleitern vorgegebenem Schema vor. Erste besondere Charaktere stellten sich heraus. Die Projektziele wurden nochmals vorgegeben, und es wurde betont, daß dieses Projekt zum größten Teil selbstorganisiert sein soll. Man verabredete sich wieder für die nächste Woche.
Ebenfalls zum Semesterbeginn fand ein Kolloqiumsvortrag von Frank Mittelbach zum Thema „Kann automatisierter Satz den Handsatz ersetzen?“ statt. Einige Projektteilnehmer besuchten den Vortrag, in dem Herr Mittelbach Beispiele für schlechten Textsatz gab. Die Begriffe „Schusterjunge“ und „Hurenkind“ zogen ins Projekt ein. Zudem wurde TEX als projektrelevant erkannt, das nach Herrn Mittelbach immer noch das beste Ergebnis bei automatisiertem Formatieren eines Textes liefert.
Bei dem ersten Wochenendseminar in Syke vom 18.–19.11.92 wollten wir unser weiteres Vorgehen festlegen. Dazu sahen wir uns bestehende Textverarbeitungssysteme an, um neue Funktionalitäten formulieren zu können, die noch kein vorheriges System anbietet. Arbeitsgruppen wurden gebildet, die sich einen Teilaspekt unseres Systems erarbeiten und als Anforderung formulieren sollten. Zwei Vorträge zu Programmiersprachen (SmallTalk-80, ASpecT), die bei einer späteren Implementierung zur Auswahl stehen würden, starteten eine Diskussion um die zu verwendende Programmiersprache, die erst Mitte des dritten Projektsemesters endete.
Am 10.1.93 luden ausländische Projektteilnehmer das Plenum zu einem chinesischen Essen ein. Immerhin waren mehr als 20 Leute zu bewirten, und unsere ausländischen Freunde nahmen viel Mühe auf sich, so daß dieser Abend wirklich großartig war.
Zum Semesterende faßten alle Arbeitsgruppen ihre bisherigen Ergebnisse zusammen. Die Einzelergebnisse wurden in einer Anforderungsdefinition gesammelt [For93]. Diese nahmen wir als Grundlage für unser weiteres Vorgehen.

2.2 2. Semester April '93 – Sept. '93

Wir starteten ins zweite Semester mit einem weiteren Wochenendseminar in Syke. Nachdem wir eine Anforderungsdefinition im ersten Semester für unser System erstellt hatten, kamen die ersten Streichungen. Die Anforderungsdefinition beschrieb ein fiktives System, das von seiner Funktionalität her in dem Projektzeitraum nicht zu implementieren war. Zuviele neue Funktionalitäten waren darin beschrieben, die in den bisherigen Textverarbeitungssystemen nicht vorhanden waren. Wir setzen Prioritäten, um eine Systemanforderung zu erhalten, die in der verbleibenden Zeit zu implementieren war, und die dabei noch über die übliche Funktionalität bestehender Textsysteme hinausgehen sollte. Weiterhin wurde die Programmiersprachendiskussion mit zwei Beiträgen zu C++ und ASpecT wieder belebt.
Im Zuge der Recherche in den Arbeitsgruppen trafen wir auf eine Dokumentenbeschreibungssprache, die von IBM entwickelt wurde. SGML[1], so ihr Name, beschreibt die logische Struktur eines Dokumentes. Somit war SGML auch für unser System von Bedeutung. Am 11.5.93 machte sich eine kleine Gesandtschaft nach Rotterdam auf, um an einer SGML-Tagung teilzunehmen. Sie konnten aus Textverarbeitungssystemen, die mit SGML arbeiteten, Anregungen für unser System übernehmen. Zudem wurde deutlich, daß es nur wenige interaktive Systeme mit WYSIWYG gab, die Dokumente mit Rücksicht auf ihren logischen Aufbau formatierten. Mit unserem System beschritten wir einen Weg, den nur wenige vorher gegangen waren.
Am 25.6.93, dem Tag des Fußballturniers des Fachbereichs 3, lief auch eine Mannschaft aus ForauSianern auf. Durch lautstarke Unterstützung aus der Fan-Kurve[2] und gutem Laufzeitverhalten der Spieler konnte ein hervorragender Platz unter „ferner liefen“ erspielt werden.
Zum Ende dieses Semesters wurde von jeder Arbeitsgruppe ein Bericht verfaßt, der die Ergebnisse ihrer Arbeit festhielt.

2.3 3. Semester Okt. '93 – März '94

Zu Beginn dieses Semesters hatten wir eine Anforderungsdefinition vorliegen, die ein System beschrieb, das wir zu implementieren gedachten. Dieses Dokument entstand aus der abgespeckten Anforderungsdefinition aus dem ersten Semester und den Berichten der Arbeitsgruppen aus dem zweiten Semester.
Die Projektleitung hatte bisher kaum in den Projektverlauf eingegriffen und überließ es der Organisationsgruppe, das Projekt zu führen. Doch durch das etwas magere Ergebnis des zweiten Projektsemesters und unserer Ziellosigkeit zu Beginn des dritten Semesters sah sie sich gezwungen, uns mit Meilensteinen, die die Implementierungsschritte bis zum lauffähigen System an Termine banden, auf die Sprünge zu helfen.
Zum Erreichen des ersten Meilensteins, einem lauffähigen Prototypen, wurden die alten Arbeitsgruppen aufgelöst. Fünf neue Arbeitsgruppen entstanden, die einzelne Module zu implementieren hatten. Die Diskussion um die zu verwendende Programmiersprache fand ein Ende in der äußerst knappen Wahl von ASpecT [vHS93]. Die Implementierung konnte beginnen.
Die Projektkasse war gefüllt. Der niedrige Stand der Zinsen regte nicht zum Sparen an. Somit wurde am 15.12.93 im Zuge einer Projektkohlfahrt das Geld der Wirtschaft wieder zugeführt.
Bis zum Ende dieses Semesters war die Projektarbeit bestimmt durch Hacken von ASpecT-Code. Die Implementierungsphase hatte begonnen.

[1] **S**tandard **G**eneralized **M**arkup **L**anguage, s. [Gol90]
[2] Schlachtruf: „Volle Kraft ForauS!“

2.4 4. Semester April '94 – Sept. '94

Das ganze Semester durch wurde implementiert. Nachdem erste grundlegende Module fertiggestellt wurden, wuchs die Funktionalität des Programmes fast wöchentlich. Doch es zeichneten sich Probleme ab, die bei Beginn der Implementierung nicht bedacht worden waren. Umstellungen in einzelnen Modulen wurden vorgenommen, so daß dort Mehrarbeit entstand.

Eine Delegation von uns besuchte die Tagung „SGML in der Praxis" in Heidelberg [Fac94]. Referenten aus der Industrie berichteten von ihren Erfahrungen mit SGML. Führende Großunternehmen wie BMW, Lufthansa und MBB verwenden SGML in den hausinternen Publikationen. So verfaßt Lufthansa die technische Dokumentation zu ihren Flugzeugen in SGML. Zudem gibt es dort eine Vorschrift, alle sonstigen firmenbezogenen Dokumente ebenfalls mit SGML auszuzeichnen. Ein System, wie wir es implementierten, traf dort auf großes Interesse. Ein echter Motivationsschub, wurden wir doch von anderen Projekten an der Uni als diejenigen verschrien, die das Rad neu erfanden.

Während eines Tagesseminars in Osterholz-Scharmbeck am 5.5.94 rekapitulierten wir, was wir bereits geschafft hatten und was noch zu erledigen sei. Es war abzusehen, daß in der verbleibenden Zeit nicht die Funktionalität implementiert werden konnte, die wir unserem Programm am Anfang des dritten Projektsemesters einmal zugedacht hatten. Doch wollten wir die Zeit bis Projektschluß noch nutzen, um ein ausgewogenes Ende zu finden.

Am 18.6.94 veranstaltete die Uni Bremen einen Tag der Forschung. Interessierte Außenstehende konnten sich den universitären Forschungsbetrieb ansehen. Auch $\mathrm{F^{or}a^{u}S}$ war hier vertreten. Allerdings nur als beispielhafte Anwendung von daVinci [FW95], das auf einem Stand auf der Messe gezeigt wurde.

Der Projekttag am 7.7.94, an dem die studentischen Projekte die Ergebnisse ihrer Arbeit vorstellten, war die Geburtsstunde von $\mathrm{F^{or}a^{u}S}$-Trek. In Anlehnung an die Fernsehserie „Star-Trek" stellten wir unsere Arbeit der letzten vier Semestern vor. Dabei war es nicht unsere Intention, einen pädagogischen Effekt bei dem Publikum zu erzielen. Vielmehr ging es darum, „Schenkelklopfer" unters Volk zu bringen. Die gleiche Absicht verfolgten auch die anderen Projektvorträge, so daß die Auftritte den gewünschten Lacherfolg brachten. Zudem hatten wir einen Stand aufgebaut, an dem wir Info-Blätter verteilten und eine Demonstration unseres Programmes zeigten.

Vier Projektsemester sind zu Ende. Wir haben viel geschafft, doch nicht alles, was wir uns vorgenommen hatten. Unser Programm hat immer noch den Charakter eines Prototyps. Doch wird im Zuge von Diplomarbeiten weiter daran gearbeitet, so daß die sich daraus ergebende Version auch den interessierten Kreisen zu Verfügung gestellt werden kann.

Somit ist unsere Mission in diesem Quadranten erledigt und wir machen uns in die Weiten des Alls auf, um neue Abenteuer zu suchen, zu finden und zu erleben. End Transmission.

3. Grundlagen

3.1 Logische Struktur

Jens Warnken, Michael Jäschke (Kap. 3.1.3.1)

In dem Projekt ForauS haben wir ein neu geartetes System zur Textverarbeitung entwickelt. Dazu sollten die in den bereits existierenden Textverarbeitungssystemen enthaltenen Aspekte mit den Techniken aus dem Übersetzerbau und den Möglichkeiten grafischer Benutzungsoberflächen kombiniert werden. Während eines Wochenendseminars sammelten wir durch ein Brainstorming Ideen, die einem solchen System zugrunde liegen müssten. Dabei kamen wir auf die besondere Bedeutung der logischen Struktur eines Dokumentes und deren Auswirkung für die Bearbeitung und Formatierung des Dokumentes.

Bisherige Textverarbeitungssysteme sahen in einem Dokument nicht mehr als einen sequentiellen Fluß von Buchstabenfolgen. Dabei kann jeder Leser eines Dokumentes in diesem eine Gliederung ersehen. Jedes Dokument beinhaltet Elemente wie Kopf-, Fußzeile, Titel, Kapitelüberschriften, Absätze etc. Von den Textverarbeitungssystemen wurde diese Struktur eines Dokumentes allerdings nicht beachtet. Der Benutzer mußte die Elemente eines Dokumentes als solche sichtbar machen, indem er sie entsprechend formatierte. Titel sind zentriert und fett. Absätze sind durch Leerzeilen voneinander getrennt. In einem System, das die Struktur eines Dokumentes erfaßt, wird die sonst vom Benutzer vorgenommene Formatierung nun vom System automatisiert. Der Benutzer muß in seinem Dokument einen Titel nur als solchen auszeichnen. Das System setzt dieses Element zentriert und fett. Die Formatierung erfolgt der Struktur entsprechend und muß nicht mühsam vom Benutzer vorgenommen werden.

Um diesen neuen Ansatz weiter zu verfolgen, wurde neben anderen Arbeitsgruppen die Gruppe „Logische Struktur" gebildet. Diese sollte sich näher mit der logischen Struktur eines Dokumentes befassen. Dabei sollten folgende Fragen beantwortet werden:

- Welche generellen Anforderungen an die logische Struktur existieren?
- Welche logischen Elemente existieren in einem Dokument?
- Gibt es eine Sprache, mit der sich die logische Struktur eines Dokumentes beschreiben läßt?
- Wie unterstützt die logische Struktur die Formatierung eines Dokumentes?

Die Arbeitsgruppe arbeitete die beiden ersten Projektsemester, um obige Fragen beantworten zu können. Die Ergebnisse und der Arbeitsprozeß, der zu den Antworten führte, sollen hier aufgezeigt werden.[1]

[1] Die Ergebnisse des ersten Projektsemesters können auch in [For93] nachgelesen werden.

3.1.1 Generelle Anforderungen an die logische Struktur

Bevor wir Anforderungen definieren können, muß erst der Kontext erfaßt werden, in dem die logische Struktur zu sehen ist. Die Sicht auf ein Dokument als eine Komposition von logischen Elementen ist eine mögliche Repräsentation. Neben dieser gibt es eine physikalische Repräsentation, die das konkrete Aussehen des Dokumentes nach dem Formatieren meint. Zwischen diesen beiden Darstellungen gibt es Transformationen. Dieser Vorgang ist das zentrale Element in Textsystemen. Bei Richard Furuta fanden wir ein Modell, das diesen Vorgang näher erläutert [AFQ88].

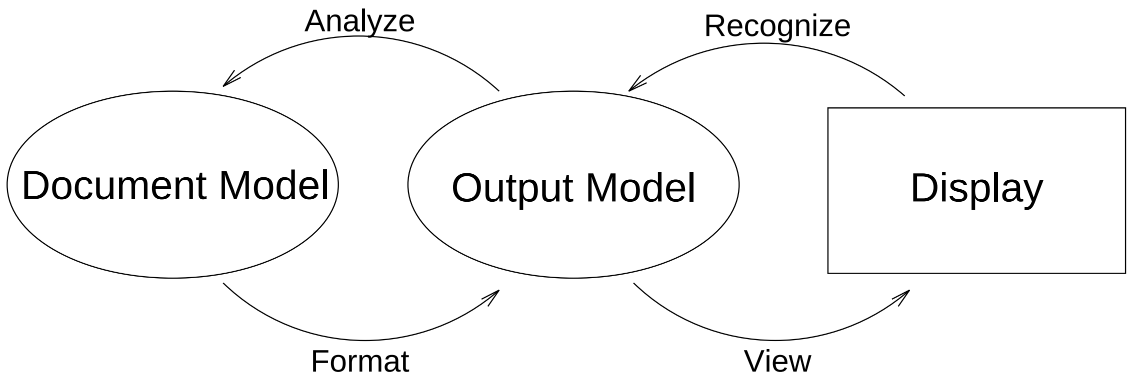

Abbildung 3.1: Repräsentations-Modell

In dem Modell sind drei mögliche Repräsentationen für ein Dokument angegeben (Abb. 3.1). Das „Document Model“ stellt eine Abstraktion auf der Ebene einer logischen Beschreibung dar. Damit ist die Form eines Dokumentes gemeint, mit der sich die Arbeitsgruppe zu befassen hatte. Durch eine Formatierung entsteht aus dem „Document Model“ das „Output Model“. In dieser Repräsentation sind die Darstellungsattribute wie Schriftart, -größe und -schnitt, Zeilen- und Seitenumbruch gesetzt. Durch eine weitere Transformation erhält man die „Display“-Repräsentation. In dieser Form hat das Dokument sein endgültiges Erscheinungsbild, wie es auf Bildschirm oder Papier dargestellt wird.
Die inverse Transformation von der „Display“-Repräsentation zum „Output Model“ (im Modell als „Recognize“ bezeichnet) beschreibt das Erkennen von Text aufgrund einer physikalischen Vorlage („Optical Character Recognition“). Dies ist für unser System ohne Bedeutung. Dagegen ist die Transformation vom „Output Model“ zum „Document Model“ (im Modell als „Analyze“ bezeichnet) für unser System von großer Bedeutung. Während einer interaktiven Veränderung am formatierten Text müssen die Änderungen auch in der logischen Struktur protokolliert werden.
Mit diesem Modell ist es nun möglich, generelle Anforderungen an die logische Struktur zu formulieren:

- Die logische Repräsentation des Dokumentes ist von der physikalischen Repräsentation zu trennen. In der logischen Beschreibung wird nur der Aufbau des Dokumentes beschrieben, nicht seine Formatierung. Es gibt eine Verbindung zwischen logischen Elementen und Formatierungsanweisungen. Allerdings ist diese nicht in der logischen Struktur erkennbar.

- Um den logischen Aufbau eines Dokumentes beschreiben zu können, müssen sich die logischen Elemente eines Dokumentes in eine hierarchische Ordnung bringen lassen. Durch diese Ordnung wird aufgezeigt, aus welchen logischen Elementen sich übergeordnete Elemente bilden. Das logische Element „Kapitel“ setzt sich z.B. aus Kapitelüberschrift und Absätzen zusammen. Auf diese Ordnung können dann Formatier- und Editierfunktionen wie das

Vererben von Formatierungsattributen an untergeordnete Elemente oder das Verschieben logischer Elemente realisiert werden. Mit dieser Ordnung lassen sich auch Dokumentenklassen definieren, die eine bestimmte Hierarchie von logischen Elementen für eine bestimmte Art von Dokumenten festlegt (z.B. die Dokumentenklasse „Buch“ mit den logischen Elementen „Titel“, „Autor“, „Vorwort“ etc). Jedes Dokument muß einer Dokumentenklasse angehören und demzufolge die logischen Elemente in der Hierarchie gesetzt haben, wie es die Dokumentenklasse vorschreibt.

- Das Dokument muß als Komposition aus kleinsten adressierbaren Objekten definiert sein. Diese „atomaren Objekte“ können z.B. als Absätze, Sätze oder Buchstaben gesehen werden. Jede Definition eines logischen Elementes terminiert in solch einem atomaren Objekt.
- Neben dem hierarchischen Anordnen von Elementen muß ein Mechanismus existieren, um Objekte miteinander zu verbinden, die nicht in einer hierarchischen Ordnung zueinander stehen. Querverweise, Fußnoten, automatisch generierte Inhaltsverzeichnisse sind Beispiele für diese Objekte.
- Es muß möglich sein, neue logische Elemente zu definieren und diese in eine bestehende hierarchische Ordnung einzugliedern. Dadurch können neben den bestehenden neue Dokumentenklassen definiert werden, die an die Erfordernisse des jeweiligen Benutzers angepaßt sind. Dabei sollen nicht neue atomare Elemente gebildet werden, sondern diese nur zur Definition des neuen logischen Elementes verwendet werden.
- Als letzte Anforderung stellen wir, daß die Sprache, in der die logische Struktur eines Dokumentes beschrieben ist, möglichst einfach gehalten ist. Somit ist das Arbeiten mit dieser Beschreibungssprache auch für den ungeübten Benutzer zu bewerkstelligen.

3.1.2 Die logischen Elemente eines Dokumentes

Die logische Struktur eines Dokumentes ist gegeben durch die Auszeichnung seiner logischen Elemente eines Textes und der Anzeige der Relationen zwischen diesen Elementen. Um die logische Struktur eines Textes zu erfassen, müssen somit zuerst die besonderen Elemente des Textes gesucht werden. Dann ist zu klären, in welcher Abhängigkeit oder Hierarchie sie zueinander stehen.
Um die logischen Elemente zu finden, haben wir uns eine einfache Daumenregel entwickelt: die logischen Elemente sind diejenige Textteile eines Dokumentes, die eine besondere Formatierung erfahren sollen. Hierunter fallen Textsegmente wie Titel, Überschrift und ganze Absätze (wenn diese einheitlich formatiert werden). Aber auch einzelne Wörter in Absätzen sind als logische Elemente zu sehen, wenn sie eine andere Formatierung als der restliche Absatz erfahren sollen. Es kann sogar soweit ins Detail gehen, daß einzelne Buchstaben in Wörtern ausgezeichnet werden müssen, um sie besonders (z.B. größer als die Buchstaben des restlichen Wortes) zu setzen. Dies stößt bei einigen sicherlich auf Unverständnis. Allerdings gibt es unserer Meinung nach keinen besseren Ansatz, um ein Dokument in logische Elemente einzuteilen, um mit dieser Auszeichnung das Dokument zu formatieren. Für größere Segmente eines Textes wie Titel, Überschriften ist diese Einteilung intuitiv. Warum sollte dies nicht auch für die kleineren Bestandteile des Textes (Wörter, Buchstaben) gelten?
Sind einmal in mehreren Dokumenten deren logischen Elemente ausgezeichnet, wird sich die Nennung der logischen Elemente in den verschiedenen Texten wiederholen. D.h. diese Dokumente sind sich in ihrem grundsätzlichen Aufbau ähnlich. Diese Texte wollen wir zu Dokumentenklassen zusammenfassen. Dokumentenklassen zeichnen sich dadurch aus, daß sie mögliche Hierarchien zwischen den logischen Elementen eines Textes vorgeben. In einer Dokumentenklasse sind also

diejenigen Texte zusammengefaßt, deren logischen Elemente in einer vorgegebenen Reihenfolge erscheinen können. Briefe, Aufsätze, Bücher sind mögliche Dokumentenklassen. In einer Dokumentenklassenbeschreibung muß dazu die mögliche Hierarchie der logischen Elemente angegeben werden können. Wir standen also vor dem Problem, eine geeignete Sprache für das Auszeichnen der logischen Elemente eines Textes zu finden, die zudem Dokumentenklassen unterstützt.

3.1.3 Beschreibungssprachen der logischen Struktur

Anfangs gingen wir in der Arbeitsgruppe davon aus, selbst solch eine Beschreibungssprache entwickeln zu müssen. Doch trafen wir während der Arbeit auf mehrere Standards, aus denen wir schließlich einen für unser System auswählten. Die einzelnen Standards sollen hier kurz vorgestellt werden, um unsere Wahl zu begründen.

3.1.3.1 ODA - Office Document Architecture

Bei ODA handelt es sich um ein genormtes Austauschformat für Textverarbeitungssysteme, welches neben reiner Text- auch Bildverarbeitung zuläßt. Entworfen wurde es erstmalig von der European Computer Manufacturers' Association und ging später in die ISO ein. ODA ist nicht dafür gedacht, direkt vom Benutzer verwendet zu werden. Es soll intern angewendet werden, zum Beispiel auf Print- und Fileservern.

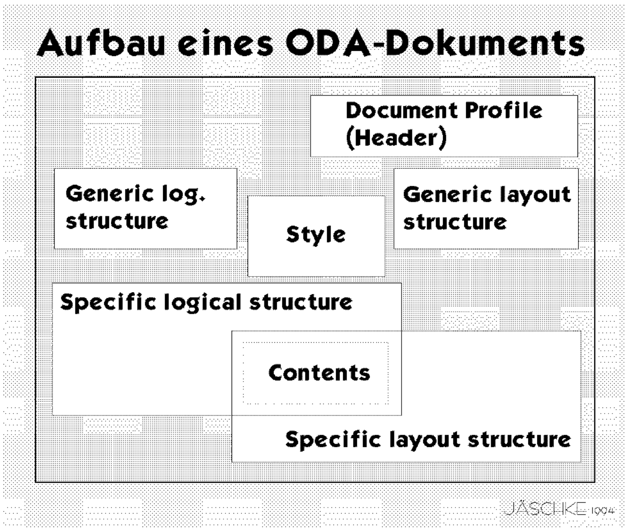

Abbildung 3.2: Graphische Darstellung des Aufbaus eines ODA-Dokuments.

Ein ODA-Dokument gliedert sich grob gesehen in zwei Teile, der *logical structure* und der *layout structure*. Beide sind allerdings miteinander verwoben. Außerdem muß innerhalb dieser Teile noch zwischen der *generic structure* und der *specific structure* unterschieden werden. Der Unterschied ist einfach erklärt. Mal angenommen, es gäbe zwei Aufsätze A1 und A2. Aufsatz A1 enthält eine Überschrift, drei Absätze und eine Zusammenfassung. Dies ist dann die spezifische logische Strukturbeschreibung für A1. Weiterhin angenommen es gäbe einen Aufsatz A2 der ebenfalls eine Überschrift und eine Zusammenfassung enthält, aber vier Absätze besitzt. A1 und A2 besitzen dann zwei unterschiedliche spezifische logische Strukturen, die aber beide auf dem gleichen Modell

basieren, der *generic structure*. Es können somit Klassen von Dokumenten unterschieden werden, die alle auf der gleichen typischen Struktur basieren.
Die logische Struktur enthält also Information darüber, wie die Teile eines Dokuments (Absätze usw.) logisch zusammengefügt werden. In der Layoutstruktur dagegen wird festgelegt, wie die einzelnen Teile darzustellen sind. Sie enthält zum Beispiel die Position für die Überschrift auf einer Seite, deren Schriftgröße und -art.
Alle bisher genannten Strukturen sind hierarchisch aufgebaut. So gibt es als kleinste Einheit die sogenannte *basic object definition*. In ihr können beispielsweise Wörter oder Zahlen enthalten sein. Die übergeordnete Einheit ist dann ein zusammengesetztes Objekt, die *composit object definition*, welche neben *basic object definitions* auch andere *composit object definitions* enthalten kann.

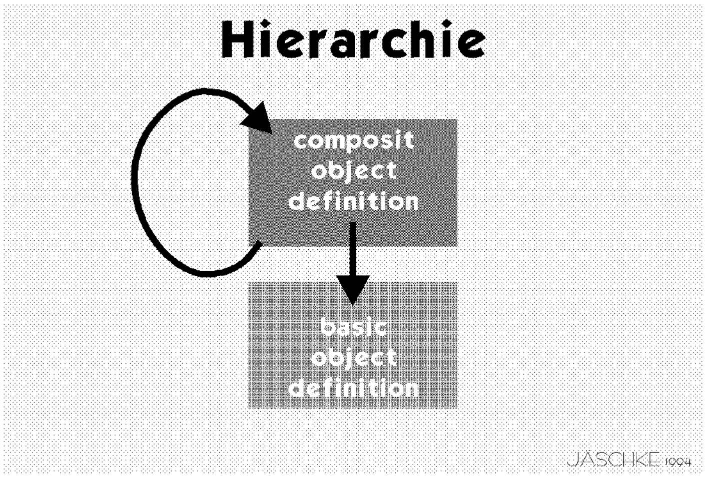

Abbildung 3.3: Hierarchischer Aufbau der ODA-Strukturen.

Wir haben uns sehr bald gegen ODA entschieden, da es einen für uns signifikanten Nachteil besitzt — Layout und logische Struktur sind nicht sauber voneinander getrennt. Da es unser Ziel ist, mehrere Sichten auf ein Dokument darstellen zu können, ist aber eine Trennung der beiden Strukturen unabdingbar. Die in ODA aus Performancegründen gewählte Verquickung der Strukturen wirkt sich bei uns eher nachteilig aus.

3.1.3.2 GRIF

Richard Furuta beschrieb in [AFQ88] ein Textverarbeitungssystem, das eine interaktive Bearbeitung von strukturierten Dokumenten mit WYSIWYG anbietet. Somit bot dieses System die Funktionalität an, die wir auch in unserem System verwirklichen wollten. GRIF, so der Name dieses Systems, und die in diesem System verwendeten Beschreibungssprachen stellen zwar keinen Standard dar, sind aber von der Idee und Verwirklichung her sehr mit unserem System verwandt. Zudem unterstützt GRIF einen Standard, den wir unserem System zugrunde legten, so daß eine Abhandlung an dieser Stelle angebracht ist.
GRIF wurde im Jahre 1984 von Irène Vatton und Vincent Quint von dem Laboratoire de Génie Informatique an der Universität Grenoble entwickelt [QV86]. Die erste Implementation erfolgte auf einer Perq unter Unix, doch gibt es jetzt Versionen für Sun-4 unter X-Windows und für den PC mit Windows.
GRIF unterscheidet bei einem Dokument dessen logische Struktur und deren Darstellung in Form

der Formatierung. Es wird also die logische Repräsentation des Dokumentes von seiner physikalischen Repräsentation getrennt, wie wir es auch als Anforderung an unsere eigene Beschreibungssprache gefordert haben. Um die logische Struktur beschreiben zu können, haben Vatton und Quint eine eigene Pascal-ähnliche Sprache entwickelt (im System als „Structure Schema" betitelt). Dieses „Structure Schema" spezifiziert Sequenzen von den logischen Elementen eines Dokumentes. Jedes logische Element (z.B. „Absatz") wird durch in den Schlüsselwörtern `begin` und `end` eingeschlossenen untergeordneten Elementen definiert. Dabei können die untergeordneten Elemente ein- bis mehrmals im übergeordneten Element vorkommen (durch das Schlüsselwort `list of` angezeigt). Zudem gibt es eine Auswahl aus einer Sammlung untergeordneter Elemente (`case of`).
Um aus dem „Structure Schema" eine Darstellung auf den Bildschirm zu erreichen, wurde eine weitere Beschreibungssprache entworfen. Dieses „Presentation Model" gibt Regeln an, wie jedes einzelne logische Element aus dem „Structure Schema" zu formatieren ist. Dazu werden die typografischen Attribute wie Schriftart, -größe, -schnitt und Position zu jedem logischen Element gesetzt. Daneben können neue Elemente, sogenannte „Presentation Elements", definiert werden. Diese Elemente sind nicht in der logischen Struktur des Dokumentes verankert. Sie erscheinen aber in dem formatierten Dokument. Vom System vergebene Nummern zum Auszeichnen von Kapiteln, Tabellen, Bildern oder Fußnoten fallen in diese Kategorie. Aber auch Textergänzungen wie „Inhaltsverzeichnis", „Tabellenverzeichnis" oder grafische Elemente wie der Querstrich bei Fußnoten stellen „Presentation Elements" dar.
Während der Bearbeitung eines Dokumentes erlaubt es GRIF, mehrere Sichten auf ein Dokument gleichzeitig zu öffnen. So kann es eine Gliederungssicht geben, in der nur die Kapitelüberschriften angezeigt sind. Gleichzeitig kann eine Sicht mit dem Volltext auf dem Bildschirm stehen. Bei Änderungen in einer der Sichten werden die anderen Sichten aktualisiert.
Dieses System bot somit eine Fülle von Anregungen, die auch zum Teil in unser System eingeflossen sind. Doch was bei GRIF unsere besondere Aufmerksamkeit erfuhr, war die Unterstützung einer standardisierten Dokumentenbeschreibungssprache names SGML. So ist es in GRIF möglich, das „Structure Schema" in einer SGML-Notation zu speichern. Beim näheren Studium des SGML-Standards fanden wir alle unsere in Kap. 3.1.1 gestellten Anforderungen erfüllt.

3.1.3.3 SGML

Das Kürzel SGML steht für Standard Generalized Markup Language [Gol90]. SGML ist ein ISO-Standard (Nr. 8879), der 1981 von IBM initiiert wurde.
SGML wurde entwickelt um das Hauptanwendungsgebiet des Computers, die Textverarbeitung, weiter zu fördern. Dabei ist unter Textverarbeitung nicht nur das Schreiben und Formatieren des Textes zu verstehen. Vielmehr schließt es auch das spätere Bearbeiten wie Stichwortsuche, Umformatieren, Datenlagerung etc. ein. Bei bisherigen Textsystemen war nach einer Bearbeitung im Sinne einer Formatierung der Text mit Formatierungsanweisungen durchsetzt, die keinem Standard entsprachen. Jedes Textsystem hat seine eigenen Konstrukte, um eine Zeile als fett und zentriert auszuzeichnen. Diese Texte waren für eine spätere Bearbeitung nicht mehr geeignet, da in die eigentlich sinngebenden Textteile die Formatierungsanweisungen in oftmals kryptischer Form eingeflochten waren. Solch ein Text konnte nur mit dem Textsystem nochmals bearbeitet werden, mit dem es ursprünglich geschrieben wurde. Der Text war nicht portierbar.
SGML bietet Konstrukte an, um einen Text für jegliche Art der Bearbeitung auszuzeichnen. Je nach Bearbeitung muß eine entsprechende Interpretation der Auszeichnung erfolgen. Somit ist ein mit SGML ausgezeichneter Text beliebig bearbeitbar. Und da SGML einen Standard darstellt, sind diese Texte von jedem standardkundigen interpretierbar.

Bei SGML handelt es sich um eine Auszeichnunssprache (im Fachjargon „markup language"). D.h. zu dem eigentlichen Text, der mit SGML bearbeitet werden soll, kommen noch Auszeichnungen („markups"). Die Auszeichnungen spezifizieren die für die Bearbeitung eines Textes wichtigen Elemente. Dies kann im Falle einer Datenbankanbindung das Adreßfeld eines Briefes sein. In unserem Fall würden in SGML die logischen Elemente ausgezeichnet werden.
SGML erfaßt die logische Struktur eines Dokumentes. Dazu gehört jedes SGML-Dokument einer Dokumentenklasse an. Die Dokumentenklassen werden in einer eigenständigen Notation definiert, der Document Type Definition (im folgenden DTD). Eine DTD nennt jedes auszuzeichnende Element aus dem SGML-Text und deklariert seinen Aufbau. Eine Elementdeklaration besteht aus untergeordneten, auszuzeichnenden Elementen und verschiedenen Konstrukten der folgenden Art:

- Das mehrmalige Wiederholen eines untergeordneten Elements (z.B. definiere sich das Element „Kapitel" durch das mehrmalige Vorkommen des Elementes „Absatz").
- Die Sequenz untergeordneter Elemente (z.B. definiere sich das Element „Aufsatz" durch die Sequenz der Elemente „Einleitung", „Hauptteil" und „Eigene Meinung").
- Die beliebige Reihenfolge untergeordneter Elemente (z.B. definiere sich das Element „Kapitel" durch die beliebige Reihenfolge von den Elementen „Absatz" und „Bilder").

Im SGML-Standard sind vier verschiedene atomare Objekte definiert, die in den Elementdeklarationen verwendet werden können.
Zu den Elementdeklarationen kommen noch Attributdeklarationen, mit denen sich an die auszuzeichnenden Elemente Attribute unterschiedlichster Art binden lassen. Diese Attribute können bei Bearbeitung des Textes ausgewertet werden.
Die Komplexität des SGML-Standards läßt sich hier schon erahnen. Zudem wird deutlich, daß SGML weitaus mehr Funktionalität anbietet, als wir zu verwenden dachten. Unsere Forderungen an eine Beschreibungssprache für die logische Struktur erfüllt SGML allemal. Zudem findet SGML auch in der Industrie immer stärkeren Einsatz. Die Lufthansa verfaßt die technischen Beschreibungen ihrer Flugzeuge in SGML. Bei BMW wird zudem die unternehmensinterne Post in SGML geschrieben. Die Entscheidung, SGML in unser System zu integrieren, war die logische Schlußfolgerung.

3.1.4 Die logische Struktur und die Formatierung eines Dokumentes

Die DTD gibt die Dokumentenklasse an, der ein SGML-Text angehört. Der SGML-Text selbst mit Auszeichnungen stellt das eigentliche Dokument dar. Diese beiden Quelltexte reichen noch nicht aus, um zum formatierten Endprodukt zu gelangen.
Wie in Kap. 3.1.2 schon erwähnt, ist jedes besonders formatierte Element eines Textes auch als logisches Element zu verstehen. Im einfachsten Fall ist der gesamte Text in einheitlicher Schriftart, -größe und -schnitt gesetzt. Dann bräuchte man nur ein logisches Element zu deklarieren, das den gesamten Text beinhaltet. Sobald Textteile durch eine andere Formatierung hervorzuheben sind, müssen diese auch mit einem logischen Element ausgezeichnet werden. Dabei kann die Formatierung eine andere Schriftart, -größe und -schnitt bedeuten. Doch auch Fußnoten, Kopf- und Fußzeilen, geschützte Leerzeichen, Verzeichnisse oder mehrspaltiger Text stellen besonders formatierten Text dar. Sie sind somit als eigene logische Elemente zu kennzeichnen.
Der Vorteil dieses restriktiven Vorgehens ist darin zu sehen, daß die eigentlichen Formatierungsregeln vom SGML-Text gelöst werden können. Um zu beschreiben, wie ein SGML-Text zu formatieren ist, muß noch jedem logischen Element, wie es in der DTD definiert ist, sogenannte

„Präsentationsregeln" (PR im folgenden) zugeordnet werden. Die PR geben an, wie ein logisches Element (d.h. sein ihm inneliegender Text) typografisch zu behandeln ist, also in welcher Schriftart, -größe oder -schnitt es zu setzen ist, sowie dessen Position auf der Seite, ob es numeriert ist (z.B. das logische Element Kapitel) etc. Die PR können vom Benutzer geändert werden, so daß er die Formatierung seines Textes beeinflussen kann, ohne den Text neu auszeichnen zu müssen. Die PR müssen sehr komplex gestaltet sein, um den Ansprüchen guter Typographie zu genügen. Es müssen sich alle im Satz üblichen Formatierungen beschreiben lassen. Dazu gehören triviale Sachen wie schon mehrmals erwähnte Schriftart, -größe oder -schnitt. Aber auch komplexere Formatierungen wie mehrspaltiger Text, mitfließende, eingebundene Objekte, geschützte Leerzeichen, verschiedenartige Verzeichnisse (Kapitel, Grafiken, Quellen), Numerierungen (Kapitel, eingebundene Objekte, Seiten), Fußnoten, Kopf-, Fußzeilen müssen sich so beschreiben lassen, daß sie ohne zu großen Aufwand erstellt bzw. geändert werden können.
Dazu konnten wir wieder auf einen Standard zurückgreifen, der im Zusammenhang mit SGML entwickelt wurde. DSSSL steht für Document Style Semantics and Specification Language (ISO-Standard Nr.10179) [ISO91]. Die Sprache wurde für die Spezifikation einer Dokumentenbearbeitung in Form der Formatierung oder anderer Datenverarbeitung entwickelt. DSSSL folgt damit dem Konzept von SGML, das ebenfalls nicht nur für die reine Formatierung eines Dokumentes entwickelt wurde. Der DSSSL-Standard ist so mächtig gehalten, um damit jedwegige Form der Dokumentenbearbeitung zu ermöglichen. So ist die Transformation eines Dokumentes in eine Datenbanksprache (SQL) vorgesehen. Da DSSSL für unsere Ansprüche zu mächtig war und die Standardisierung noch nicht abgeschlossen ist, haben wir uns entschlossen, diesen Standard nicht zu unterstützen. Damit oblag es uns, eine eigene Sprache, die die PR zu den logischen Elementen angibt, zu entwickeln. Dazu gab es eine eigenständige Arbeitsgruppe, deren Bericht in Kap. 3.2 nachzulesen ist.
Wir haben damit eine klare Dreifachteilung eines in unserem System zu bearbeitenden Textes gefunden. Als erstes der in SGML-Notation in logische Elemente eingeteilte Quelltext. Zweitens die dazugehörige DTD und zuletzt die PR. Zu jedem SGML-Text gehört eine DTD, die angibt welche Elemente in dem SGML-Text ausgezeichnet werden können, und in welcher Reihenfolge sie erscheinen. Zu jeder DTD gehören die PR, die angeben, wie die in der DTD spezifizierten Elemente zu formatieren sind.

3.2 Präsentationsregeln

Gunnar Kaveman

Zentrale Aufgabe der Präsentationsregeln PR ist die Verknüpfung der logischen Elemente eines Dokumentes mit deren tatsächlicher physikalischer Erscheinung auf dem Bildschirm oder als gedruckter Text auf dem Papier. Schon bei der Entwicklung der logischen Struktur für das ForauS–System hatten wir festgelegt, daß es möglich sein müsse, den gleichen logischen Elementen des Dokumentes für verschiedene Sichten ein unterschiedliches Aussehen und damit einen anderen Kontext zu geben. Dies ist die Aufgabe der Präsentationsregeln. Bei der Entwicklung der logischen Struktur hatten wir uns bereits mit einigen Ansätzen für die Gestaltung der Präsentationsregeln beschäftigt, jedoch noch keine endgültigen Festlegungen getroffen.
Einem Plenumsbeschluß entsprechend wurden verschiedene Arbeitsgruppen gebildet, die jede für sich die Aufgabe hatten, innerhalb von drei Wochen ein Konzept für die Präsentationsregeln zu entwickeln, und dieses auf dem Treffen des Projektes vorzustellen.
Nachdem alle Gruppen ihre Konzepte vorgestellt hatten, wurde beschlossen, aus einigen Mitgliedern der drei Gruppen mit den gelungensten Konzepten eine neue Gruppe zu bilden. Diese

Gruppe bekam den Auftrag, ein endgültiges Präsentationsregel–Konzept zu erstellen und daran anschließend eine entsprechende PR–Sprache zu entwickeln.
Da diese Gruppe wieder neu gebildet worden war, mußte zuerst eine gemeinsame Wissensbasis hergestellt werden. Das geschah durch die intensive Diskussion und Vorstellung der verschiedenen Konzepte und durch die nochmalige Lektüre der gesamten vorhandenen Aufsätze und Artikel zu diesem Thema.
Zuerst wurde dann ein gemeinsames Grobkonzept geschaffen mit dem sich die folgenden Anforderungen realisieren ließen:

- Die Elemente können einzeln ausgezeichnet werden. Schriftart, –größe und –schnitt können festgelegt werden.
- Die Elemente können in beliebiger Reihenfolge zu neuen Elementen zusammengefaßt werden.
- Die Elemente müssen beliebig hintereinander und nebeneinander auf einer Seite positioniert werden können.
- Das Seitenlayout muß definiert werden können.
- Die Auszeichnungen können für jede gewünschte Sicht unterschiedlich angegeben werden.
- Die Attribute werden, wenn sie in den nachgeordneten Elementen nicht definiert sind, von den Vorgängern geerbt.

Wir entwickelten nun die erste Version der Sprache. Unter Berücksichtigung der Überlegungen von Brown und Blair [BB90] sowie des Ansatzes von Furuta, Quint und André [AFQ89], der auch bei der Realisierung ihres GRIF–Systemes Anwendung fand, entschieden wir uns dafür, eine blockorientierte Sprache zur Beschreibung der Auszeichnungen und Verknüpfungen zu benutzen. Die Definition der logischen Elemente ist bereits in der DTD geschehen. Für diese, in den PR als `logische Objekte` bezeichneten Teile des Dokumentes, können nun Regeln, die deren Erscheinungsbild bestimmen, in der blockorientierten PR–Sprache angegeben werden.
Die freie Positionierung der Objekte auf den Seiten stellten wir für das Grobkonzept erst einmal zurück, da wir noch kein gut realisierbares Konzept für diese Aufgabe hatten.
Für die Definition der Seitenformate lehnten wir uns an die von LaTeX benutzten und die in der Literatur [Kop89] beschriebenen Methoden an. Wir beschlossen allerdings schon frühzeitig, die Vielzahl der für das Seitenlayout möglichen Attribute zu beschränken. Uns war klar, daß damit den Designern einige Gestaltungsmöglichkeiten genommen würden, aber wir waren überzeugt, daß ein gutes Seitenlayout auch mit einer geringeren Anzahl von Attributen erreicht werden kann. Außer den ausschließlich für die Aufteilung der Ränder und Textbereiche bestimmten Parametern legten wir auch gleich die von uns für notwendig erachteten Attribute für das Absatzlayout fest.
Die verschiedenen Auszeichnungen für unterschiedliche Sichten werden durch das Konstrukt `VIEWS` ermöglicht: Bei der Beschreibung der Dokumentauszeichnung müssen die Sichten, in denen das Dokument später zu betrachten sein soll, zwingend angegeben werden. Mit dem Schlüsselwort `VIEW` kann bei der Definition der Objekte dann jedes Objekt für jede Sicht unterschiedlich ausgezeichnet werden. Wird bei der Objektdefinition keine Sicht angegeben, so gilt diese Definition global.
Für den Vererbungsmechanismus brauchten wir keine besonderen Konstrukte in der Sprache schaffen. Wir legten sie per Definition so fest: Wenn ein bestimmtes Attribut eines Objektes nicht gesetzt ist, so wird der Wert des gleichen Attributes des Elternobjektes benutzt. Ist auch in den Elternobjekten dieser Wert nicht definiert, wird der Wert aus der globalen Definition genommen.

Mit diesem ersten Entwurf waren wir nun schon weitestgehend in der Lage, Dokumente in dem Umfang auszuzeichnen, den wir anstrebten. Inzwischen hatten auch die Arbeitsgruppen, die sich mit der Formatierung befassten, ihre grundlegenden Konzepte fertiggestellt. Wir setzten uns nun mit dieser Gruppe in Verbindung, um unser Grobkonzept einmal von denen, die es ebenfalls betraf, begutachten zu lassen und um zu erfahren, ob noch weitere oder andere Parameter zur Auszeichnung und Formatierung benötigt würden.
Dabei stellten wir fest, daß es günstig wäre, wenn der Formatierer einige Objekte mit den dazugehörigen Formatierregeln bereits kennen würde. Damit wäre der Aufwand beim Formatieren relativ gering, und der Formatierer könnte unter diesem Aspekt verhältnismäßig einfach gestaltet werden. Wir entschieden uns, dabei einem Modell, das Furuta [Fur89] beschrieben hat, zu folgen: Fast alle Dokumente bestehen aus Textkomponenten, die mit verschiedenen anderen speziellen Komponenten wie zum Beispiel Formeln, Tabellen oder Graphiken vermischt sind. Zusammen mit der Formatierergruppe legten wir fest, welche Komponenten unser System kennen sollte und wie diese zu behandeln seien. Im Formatierer werden dann diese Komponenten und ihre möglichen Attribute und Parameter „fest verdrahtet“. Dies schränkt zwar theoretisch die Gestaltungsmöglichkeiten ein, erschien uns aber der beste Weg, um das System einigermaßen schnell und kompakt zu halten. Die anfängliche Menge der Formatierobjekte wurde im weiteren Verlauf noch mehrmals ergänzt beziehungsweise geändert. Die von uns endgültig in ForauS benutzten Formatierobjekte sind:

- `catalog` zum Erstellen von Verzeichnissen.
- `counter` zur Definition von Zählern.
- `index` zur Erzeugung von zum Beispiel Schlagwortverzeichnissen.
- `layout` zur Anordnung anderer Objekte.
- `note` zur Erzeugung von zum Beispiel Fußnoten.
- `reference` zum Erzeugen von Verweisen.
- `text` für einfachen Text.

Zur Unterstützung der physikalischen Repräsentation der logischen Objekte führten wir zusätzlich noch `physikalische Objekte` ein. Mit ihrer Hilfe können logische Objekte, in denen die gleichen Objekt– beziehungsweise Attributdefinitionen mehrfach benutzt werden sollen, in der gewünschten Reihenfolge angeordnet werden. Außerdem ermöglichen sie, sich wiederholende Textpassagen wie zum Beispiel Kopfzeilen, in denen Texte vorkommen, die nicht in einem logischen Element vorhanden sind, zu erzeugen. Sie können allerdings nicht direkt angezeigt werden, sondern müssen durch ein `logisches Element` benutzt werden.
Zur Lösung des Problems der freien Positionierung der Objekte auf den Seiten beschäftigten wir uns zuerst mit dem Boxenkonzept von TEX[Knu86] und dessen Erweiterungen durch die Makros von LATEX[Lam86]. Die Möglichkeiten, die mit diesen Systemen realisierbar sind, reichten für die Umsetzung unserer Vorstellungen jedoch nicht aus, beziehungsweise wäre deren Realisation nur mit erheblichem Aufwand zu erreichen gewesen. Aber einige der dort vorgestellten Ideen und der Artikel von Jobloboff [Job89] versetzten uns in die Lage, ein für unsere Zwecke brauchbares und realisierbares Konzept zu entwickeln und umzusetzen. Zusätzlich zum `position`–Attribut, mit dem die Objekte grob auf den Seiten positioniert werden können, führten wir auch noch eine Möglichkeit ein, Objekte miteinander zu verbinden. Durch das `keepwith` wird erreicht, daß der Formatierer die damit verbundenen Objekte zusammen oder auf der gleichen Seite setzen muß.

Mit der Schaffung der Positionierungsmöglichkeit war unser Konzept komplett. Wir stellten die fertige PR–Sprache im Plenum vor. Einige der in der nachfolgenden Diskussion angeregten Änderungen und Ergänzungen der durch unsere Sprache zur Verfügung gestellten Attribute und Parameter wurden noch in den Sprachumfang eingebracht. Nach zahlreichen Diskussionen mit der Parsergruppe und daraus resultierenden Änderungen an der Syntax unserer Sprache konnten wir unseren Arbeitsauftrag beenden und die endgültige Version der PR–Sprache präsentieren. Die komplette PR–Sprache ist in [PF93] ausführlich beschrieben.

3.3 Typographie und Formatierer

Tarik Ali, Andree Hähnel

3.3.1 Die Entstehung

Das erste mehrtägige Projekttreffen in Syke sollte in einem groben Rahmen die Schwerpunkte unseres Projektes abstecken. Zu diesem Zweck wurde ein Brainstorming anberaumt, um jedem die Chance zu geben vorzubringen, welche Punkte ihm bei der Entwicklung eines neuartigen Textsystems als wichtig erscheinen. So wollten wir entscheiden, in welche Richtung unsere Textverarbeitung gehen sollte.
Bei diesem Brainstorming wurden unter anderem folgende Punkte genannt:

- Dem Formatieren muß verstärkte Aufmerksamkeit zukommen!
- Was ist „schöner Text“ überhaupt?
- Wir müssen uns um Kerning und Ligaturen kümmern!

Diese Einwürfe wurden zusammengefaßt und gaben so den Anstoß zur Bildung der Gruppe „Typographie und Formatierer“.

3.3.2 Die Aufgaben und Ziele

Unsere Aufgabe war es herauszufinden, was schöner Text oder gute Formatierung ist, also welche Grundlagen es zur Bewertung von Textformatierungen gibt. Desweiteren sollten wir feststellen, welche typographischen Werkzeuge existieren, um Texte in „typographisch richtiger“ Weise zu formatieren.
Ein weiteres Ziel war es, das uns angeeignete Wissen in Vorträgen und Dokumentationen an die anderen Projektteilnehmer weiterzugeben, damit bei der späteren Implementierung jeder das benötigte Grundwissen im Bereich der Typographie hat.

3.3.3 Die Arbeit in der Gruppe

Zunächst versuchten wir herauszufinden wie ausgeprägt der Wissensstand zum Thema „Formatierung von Dokumenten“ bei den einzelnen Gruppenmitgliedern ist. Hierbei mußten wir feststellen, daß nur wenige Mitglieder über einige Erfahrungen bezüglich des Setzens von Texten verfügte. Bei den anderen beschränkte sich der bisherige Umgang mit Texten auf das Editieren dieser mit Hilfe einfacher Textverarbeitungsprogrammen.
Während der anschließenden Suche nach Literatur zum Thema „Typographie“ fanden wir Unterstützung bei der Dozentin des Kurses „Systeme zur Dokumentenerstellung“. Dieser Kurs wurde

eigens für das Projekt ForauS angeboten. Sie verwies uns auf mehrere Buchtitel, von denen wir drei in die engere Wahl aufnahmen ([BK89], [Fra93] und [Knu86]). Nach dem Durcharbeiten dieser Literatur war uns aber klar, daß wir uns bei der Gruppenarbeit hauptsächlich an dem Buch von Brüggemann-Klein [BK89] orientieren würden, da es uns dem Problem am nächstliegensten erschien. Die andere Literatur wollten wir punktuell einfließen lassen.
Das erste Nahziel, das sich die Gruppe vornahm, war die Erarbeitung einer Liste von Kriterien oder Richtlinien, die Grundlage für eine anspruchsvolle Gestaltung von Texten sind. Zu diesem Zweck stellten wir einen grob gegliederten Katalog dieser Richtlinien zusammen. Der Erfolg der Verlesung dieses Kataloges im Plenum war nur mäßig, da es sich zumeist nur um die Zusammenstellung von Schlagworten handelte, die die Gruppenmitglieder nun zwar verstanden, die aber vielen anderen Projektteilnehmern noch nicht geläufig waren. Deshalb bereiteten wir in den nächsten Wochen einen umfangreicheren Vortrag vor, den wir am 7. Januar 1993 hielten. Nach diesem Vortrag hatten die Projektteilnehmer eine bessere Vorstellung davon, was schöner Text ist und welche Möglichkeiten es beim Setzen von Dokumenten gibt. Vor allem aber wußten wir nun, was es zu vermeiden galt!
Die letzte Aufgabe unserer Gruppe war es, einen Text zu schreiben, der die Ergebnisse unserer Gruppenarbeit dokumentiert. Dieser wurde dann mit den Ergebnissen der anderen Gruppen in dem Projektsemesterergebnis zusammengeführt. Das so entstandene Dokument ist eine „Anforderungsdefinition für ein innovatives Textsystem“ [For93].

3.3.4 Das Ergebnis des ersten Semesters

3.3.4.1 Typographie

Das Ziel bei der Gestaltung eines Dokumentes muß es sein, dieses leicht lesbar zu machen. Das heißt, der Leser muß das Dokument inhaltlich verstehen und in der vom Autor intendierten Form benutzen können. Wird ein Text in einer Schrift gesetzt, die die Aufmerksamkeit des Lesers auf sich zieht, anstatt sie auf den Inhalt und die Funktion des Textes zu lenken, so hat sie ihre Aufgabe verfehlt. Eine gute Strategie beim Setzen eines Dokumentes ist es, solche Irritationen des Lesers zu vermeiden.

3.3.4.2 Formatierer

Die Aufgaben die beim Formatieren eines Dokumentes ausgeführt werden müssen, können in drei Teilbereiche aufgeteilt werden:

- Wortformatierer: Dieser setzt unter Beachtung von typographischen Regeln wie Kerning und Ligaturen Buchstaben zu Worten zusammen.
- Absatzformatierer: Dieser setzt die Worte und Zeilen eines Absatzes unter Verwendung von typographischen Richtlinien wie Ausrichtung und Einzug.
- Seitenformatierer: Dieser ordnet die Absätze auf den Seiten eines Dokumentes an. Hierbei müssen typographische Regularien für den Umgang mit z.B. Fußnoten, Schusterjungen und Hurenkindern Berücksichtigung finden.

3.3.5 Vor dem zweiten Semester

Das Bestehen der meisten Gruppen endete mit der Fertigstellung des Dokumentes „Anforderungsdefinition für ein innovatives Textsystem“ am Anfang des zweiten Semesters. Im Gegensatz hierzu wurde die Fortführung der bisherigen Tätigkeiten unserer Gruppe im Plenum so hoch bewertet, daß sogar noch andere Projektteilnehmer aus den aufgelösten Gruppen zu uns stießen, um unsere Bestrebungen im Bereich Formatierer zu unterstützen.

3.3.6 Die neuen Aufgaben und Ziele

Die neuen Aufgaben waren es, TEX-Regeln zu analysieren, einen Vergleich von TEX und dem Dokumentensystem Framemaker [Fra93] durchzuführen und basierend auf den Ergebnissen des ersten Semesters Untergruppen zu bilden, die dann parallel die Grundlagen für eine Anforderungsdefinition und eine spätere Implementierung eines Textformatierers erarbeiten sollten. Die Ausarbeitungen sollten am 27. Mai 1993 im Plenum vorgetragen werden.

3.3.7 Die Arbeit in den Gruppen

Zunächst teilten wir uns in drei Gruppen entsprechend der im letzten Semester entwickelten Modularisierung eines Formatierers in Wortformatierer, Absatzformatierer und Seitenformatierer auf.

Wortformatierer
Die Gruppe Wortformatierer hatte zunächst große Probleme, geeignete Literatur zu finden. Nach Rücksprache mit dem Plenum wurde auf das TEX-Handbook von Donald E. Knuth [Knu86] zurückgegriffen, das sich auch in diesem Bereich als Standardwerk erwiesen hat.

Absatzformatierer
Die Absatzformatierer hatten schon zu Beginn ihrer Arbeit eine große Menge an Literatur zur Verfügung. Dies waren einerseits Werke des Entwicklers von TEX Donald E. Knuth, aber andererseits auch das Buch von Brüggemann-Klein. Das Material war sehr anspruchsvoll und umfangreich. Die Aufgaben wurden wiederum aufgeteilt. In internen Gruppentreffen versuchten wir in Diskussionen zu entscheiden, was unser Absatzformatierer leisten muß, und ansatzweise auch schon, wie er es leisten muß. Die Ergebnisse der Gruppen wurden im Plenum vorgetragen und zu Papier gebracht, um später bei der Implementierung hierauf zurückgreifen zu können.

Seitenformatierer
Die Aufgaben des Seitenformatierers wurde schon recht früh manifestiert. Es stand allerdings noch nicht fest, wer die Verwaltungsaufgaben für das Formatierermodul übernimmt. Anfangs wurde auch über einen expliziten Verweisverwalter diskutiert.
Als Literatur und Quellen boten sich wieder das TEX-Handbook von Donald E. Knuth [Knu86] an, was sich ausführlich über das Umbrechen von Seiten ausläßt. Außerdem waren es meist konzeptionelle Fragen, die wir in der Kleingruppe diskutieren und lösen konnten.

3.3.8 Die Ergebnisse der neuen Gruppen

Wortformatierer
Der Wortformatierer bestimmt die Ausmaße einer Wortbox und die Position der Grundlinie innerhalb dieser Box. Als Eingabeparameter braucht er Informationen über die Schriftart und das entsprechende Wort. Der Wortformatierer berücksichtigt Ligaturen und Kerning.

Absatzformatierer
Der Absatzformatierer ist das Bindeglied zwischen Wort- und Seitenformatierer. Er wird vom Seitenformatierer angestoßen mit dem Auftrag, einen Absatz zu formatieren. Er selbst stößt den Wortformatierer an, von dem er die Ausmaße der von ihm zu setzenden Worte benötigt.
Das Problem, Worte in Zeilen aufzuteilen, ist eng mit dem Problem der Silbentrennung verbunden. Das musterbasierte Trennverfahren nach Liang ist hier eine akzeptable Lösung.

Es gibt verschiedene Verfahren einen Absatzumbruch zu implementieren. Die von Knuth entwickelte Lösung des Bewertens aller lokal optimalen Umbrüche, mit dem Ziel einen global optimalen Umbruch zu bestimmen, erscheint uns am besten, aber auch am aufwendigsten zu sein.

Seitenformatierer

Der Seitenformatierer arbeitet sehr eng mit dem Absatzformatierer zusammen. Bei der Gestaltung der Seiten eines Dokumentes muß er die Layoutparameter der Seiten berücksichtigen. Der Seitenformatierer muß die verschiedenen Objekte, die in einem Dokument vorkommen können, auf mehrere Seiten verteilen. Es können folgende Objekte vorkommen:

- Absätze
- Tabellen
- Formeln
- Figuren (Grafiken usw.)

Der Seitenformatierer kann einen *Absatz* problemlos aufsplitten, um einen Seitenumbruch zu erzielen. Alle anderen Objekte sind nur sehr schlecht oder überhaupt nicht trennbar. Weiterhin sollen beim Seitenumbruch Schusterjungen und Hurenkinder vermieden werden. Fußnoten und eine Kopf- bzw. Fußzeile soll möglich sein. Desweiteren muß er Zugriff auf das intern repräsentierte Dokument haben, und eventuell übergeordnete Verweise oder Beziehungen zwischen verschiedenen Objekten verwalten und berücksichtigen.

3.4 Advanced Features

Guido Frick, Kai Hofmann

Auf dem ForauS-Seminar wurde am 19.11.1992 beschlossen, die Arbeitsgruppe *Advanced Features* zu gründen. Im Plenum vom 26.11.1992 wurde sie durch Trinh Le, Guido Frick und Kai Hofmann gebildet.

Aufgabe der Arbeitsgruppe sollte es sein, Funktionalitäten auszuarbeiten, die in herkömmlichen Textsystemen noch nicht oder nicht ausschöpfend eingesetzt werden. Dies führte schnell zu der Notwendigkeit, eine genauere Abgrenzung des Begriffes *erweiterte* Funktionalität vorzunehmen. Handelt es sich bei einer Funktionalität um eine erweiterte, wenn noch kein Textsystem diese anbietet, oder wenn es nur zwei oder drei tun? So blieben Überschneidungen mit der Gruppe Standardfunktionalitäten nicht aus, zumal hier anfangs nicht in ausreichendem Maße Absprachen stattfanden.

Das Ergebnis der Arbeitsgruppe hing ebenfalls eng von den untersuchten Textsystemen ab, nach deren Analyse erweiterte Funktionen erarbeitet wurden. Die Grundlagen der Analyse stellten die mehrjährigen Erfahrungen der Gruppenmitglieder mit Standardprodukten zur Textbearbeitung, Editoren und Layoutprogrammen dar, mit denen Informatikstudenten im Laufe ihrer Computertätigkeit reichlich in Berührung kommen.

Diese Erfahrungen waren umso wichtiger, als von vornherein nicht auf ausgiebige Literatur zurückgegriffen werden konnte, da die zu erarbeitenden Funktionalitäten eben noch nicht als Standard etabliert und somit Gegenstand von Aufsätzen und Berichten waren.

Zusätzlich flossen Wünsche und Anregungen der Projektleitung mit in die Ausarbeitung ein (z.B. mehrere Sichten auf ein Dokument), die primär Gegenstand des Projektes sein sollten.

Während der gesamten Arbeit an den Advanced Features unterstützten sich die Gruppenmitglieder gegenseitig, so daß auch schwierige Probleme gelöst werden konnten. Hierbei spielten

natürlich auch die Treffen in der GW2 Cafeteria eine entscheidende Rolle, da hier entspannt gearbeitet werden konnte. So wurden dann auch Funktionalitäten gefunden, die bisher noch nicht oder nicht ausreichend in Textverarbeitungssystemen vorhanden sind. Hierzu gehören z.B. eine Versionsverwaltung und die Bereitstellung mehrerer Sichten auf ein Dokument.
Eine detailiertere Ausarbeitung der gefundenen Features erfolgte dann später durch die einzelnen Gruppenmitglieder.
Das Endergebnis dieser Gruppe ist in [For93] festgehalten.

3.5 Standardfunktionalitäten

Jens-Martin Brinkmann, Frank Hettling

Die Arbeitsgruppe Standardfunktionalitäten wurde mit der Zielsetzung, die Standardfunktionalitäten herkömmlicher Textverarbeitungssysteme zu untersuchen und zu dokumentieren, ins Leben gerufen. Um den Arbeitsauftrag der Arbeitsgruppe Standardfunktionalitäten zu umreißen, mußte zuerst der Begriff „Standardfunktionalität“ auf seine Bedeutung hin untersucht werden. Bei der Begriffsklärung der „Standardfunktionalitäten“ in diesem Kontext konnten wir uns auf folgende Bedeutung einigen. Standardfunktionalität bedeutet: Eine Funktion wird als Standard angesehen, wenn sie signifikant häufig in verschiedenen Textverarbeitungssystemen vertreten ist, wobei dieses „Vertreten sein“ unter der zeitlichen Perspektive der stetigen Änderung der Funktionen und ihrer Mächtigkeit im Zusammenhang mit anderen Funktionen zu betrachten ist. (Diese Definition bedeutet kurz gesagt: Funktionen werden in den Standard aufgenommen und verschwinden wieder, wenn sie durch mächtigere Funktionen im Laufe der Zeit ersetzt werden). Unter dieser Perspektive fing nun die Arbeitsgruppe „Standardfunktionalitäten“ an, Textverarbeitungssysteme zu betrachten und zu vergleichen. Um diese Arbeit zu bewerkstelligen und zu einem zufriedenstellenden Abschluß zu bringen, hatten wir uns eine geniale und einfache Vorgehensweise ausgedacht. Wir zählten die Funktionen mit Hilfe einer Tabellenkalkulation. Nun war nur noch herauszufinden, welche Funktionen die Definition als Standardfunktion erfüllte.
Die ausgewählten Textverarbeitungen unterlagen einigen Einschränkungen: Sie mußten uns verfügbar sein, sie durften keine ausschließlichen DTP-Programme sein, sie durften keine Satzprogramme (Bsp. TEX) und keine Texteditoren sein. Diese Kriterien beschnitten die Objektivität dieser Untersuchung. Es wurden bei einigen Textverarbeitungen auch nicht alle Funktionalitäten erkannt, da die total gestreßten Auswerter (Frank und Jens-Martin) sich nicht in allen Textverarbeitungen perfekt auskannten, und einige Funktionen nur in Sonderfällen überhaupt erst verfügbar waren. Von der Vielzahl der Funktionen sortierten wir die Funktionen aus, die unseren Kriterien genügten. Diese Funktionen haben wir als „Standardfunktionalität“ herkömmlicher Textverarbeitungssysteme ermittelt. Die elementarsten Funktionen seien hier nochmals kurz genannt: Einfügen eines Zeichens, Löschen eines Zeichens , Drucken und — nicht zu vergessen — das Laden und Speichern eines Textes. Das Ergebnis unserer Arbeit ist eine Auflistung aller Funktionen, die als Standard in herkömmlichen Textverarbeitungssystemen verfügbar sind. Dieses Ergebnis wurde in einen der ForauS-Plenen vorgetragen und in schriftlicher Form niedergelegt. Diese Liste ist von einer stetigen Änderung ergriffen, da jedes neue Textverarbeitungssystem neue Maßstäbe in der Funktionalität setzt. (Also schauen wir auf die Funktionalität, mit der uns ForauS noch beglücken wird.)

3.6 Ausgabe und Austauschformate

Bernd Rattey, Christian Schäfer, Kai Siegele

Die Gruppe Ausgabe und Austauschformate hatte die Aufgaben, zu erarbeiten, welche Austauschformate anderer Programme existieren und zu untersuchen, wie eine Textverarbeitung die Ausgabe auf einem Bildschirm und auf einem Drucker realisieren könnte.

3.6.1 Der Anfang

Es wurde uns schnell klar, daß sich die beiden Aufgaben gut trennen ließen. Weiterhin fiel uns auf, daß die erste Aufgabe – das Ermitteln der Austauschformate – mit völlig anderen Problemen behaftet war, als die zweite Aufgabe. Um die Arbeit in der Arbeitsgruppe effektiver zu gestalten, beschlossen wir daher, uns aufzuteilen. Kai kümmerte sich vorrangig um die Austauschformate, während Christian und Bernd das Problem der Ausgabe in Angriff nahmen.
Aufgrund der Diversität der Aufgaben und der mit ihnen verbundenen Probleme werden wir auch diesen Bericht in zwei Stränge zerschneiden, die Austauschformate und die Ausgabe.

3.6.2 Die Probleme

Da die Erfahrungen unserer Gruppe zumeist aus dem Umgang mit der DOS- und Windows-Welt stammten, waren unsere Kenntnisse sehr stark von dieser Domäne geprägt.

3.6.2.1 Austauschformate

Bei der Untersuchung des Themenkomplexes für unsere Zielplattform, dem SUN-System, hatten wir das Problem, daß aufgrund des zu Beginn des Hauptstudiums geringen Bekanntheitsgrades von Standardprodukten und deren geringe Verfügbarkeit im Fachbereich, sich keine eindeutigen Aussagen hierüber machen ließen. Daher konzentrierten wir uns neben der SUN-Welt auch auf die Formate aus der PC-Welt.

3.6.2.2 Ausgabe

Wenn man sich das Konzept anschaut, mit dem die Ausgabe bei der Benutzungsoberfläche `MS-Windows` gelöst wird, läßt sich relativ schnell feststellen, daß fast jede Grafikkarte, jeder Bildschirm und jeder Drucker von der Benutzungsoberfäche mittels eines eigenen Treibers unterstützt wird.
In der SUN-Welt existieren hingegen mehrere Benutzungsoberflächen, die hervorzuheben wären, dies sind hauptsächlich `XView` und `Motif`, die beide auf `X-Windows` aufsetzen.
Somit lag das Hauptproblem darin, diese beiden Welten nach Möglichkeit unter einen Hut zu bekommen. Weiterhin schied das Treiberkonzept schnell aus, da hinter einem solchen Konzept ein ungeheurer Aufwand zum Bereitstellen und Warten der Treiber liegt.

3.6.3 Der Fortgang

3.6.3.1 Die Austauschformate

Die Arbeit untergliederte sich in drei Teile:
Zunächst überlegten wir uns, mit welchen Programmen Daten ausgetauscht werden sollen und welche Gründe hierfür ausschlaggebend sind. Wir kamen schließlich zu dem Ergebnis, daß der Austausch mit Standardprogrammen wie Datenbanken, Tabellenkalkulationen, Grafik- und Formelsatzprogrammen sinnvoll ist. Erst das Zusammenspiel mehrerer Programme erlaubt die Bearbeitung komplexerer Aufgabenstellungen, wie sie in der täglichen Praxis vorkommen. Da die

Benutzung einer Textverarbeitung häufig eine Geschmacksfrage ist, sollte auch die Integration anderer Textverarbeitungen ermöglicht werden.
Danach suchten wir nach Formaten, die von möglichst vielen Programmen verarbeitet werden können und deshalb einen reibungslosen Austausch von Daten zwischen verschiedenen Programmen gewährleisten. Diese Formate werden wir im folgenden als Standardformate bezeichnen. Dateien, die in Standardformaten vorliegen, sollten von der ForauS-Textverarbeitung auf jeden Fall verarbeitet werden können. Standardformate gibt es in der DOS und Windows-Welt allerdings sehr wenige.
Das am weitesten verbreitete ASCII-Format, welches zur Speicherung von Texten benutzt werden könnte, hat leider den Nachteil, daß vom Programm vorgenommene Formatierungen verlorengehen. Eine sinnvolle Alternative bietet hier die Speicherung eines Textes, der mit SGML-Befehlen ausgezeichnet ist. Durch SGML (Standard Generalized Markup Language) kann ein Text in logische Elemente wie Kapitel und Überschriften eingeteilt werden. Den logischen Elementen können später Formatierungsvorschriften zugeordnet werden (siehe 3.1.3.3).
Für die Druckausgabe hat sich in der Praxis das Postscript-Format durchgesetzt. Fast alle auf dem Markt verfügbaren Drucker sind in der Lage, Dateien zu verarbeiten, die im Postscript-Format vorliegen.
Grafiken werden dagegen vorwiegend im GIF- bzw. PCX-Format abgespeichert.
Ein Programm muß für den Austausch von Fremddateien geeignete Routinen zur Verfügung stellen. Über Möglichkeiten haben wir uns schließlich ebenfalls Gedanken gemacht. Am geeignetsten erschien uns der Datenaustausch über eine Clipboard-Funktion, wie sie z.B. in Windows durch die Zwischenablage realisiert ist. Allerdings wurden auch andere Alternativen angedacht, wie Referenzen auf Fremddateien innerhalb des Dokumentes oder eine zuvor durchzuführende Konvertierung in das eigene Format. Vor- und Nachteile dieser Verfahren wurden sorgfältig gegeneinander abgewogen.

3.6.3.2 Die Ausgabe

Wir entschlossen uns, zunächst das Idealbild der Ausgabe zu erarbeiten. In diesem wurden die Geräte aufgeführt, die eine ideale Textverarbeitung mit allen Optimierungen und in höchster Qualität ansprechen sollte.
Da ein solches Idealbild für unser studentisches Projekt jedoch utopisch war, versuchten wir parallel ein kompakteres Konzept zu entwickeln. So zogen wir uns jeweils eine Zeit in unser Kämmerlein zurück, um ein solches Konzept zu entwickeln. Danach stellte jeder sein Konzept der Gruppe vor und es kam, was kommen mußte: Beide Konzepte waren grundverschieden. Nun setzte der typische Prozeß in einer solchen Gruppe ein: Jeder versuchte, sein eigenes Konzept zu verteidigen, und das des Gegenübers in Frage zu stellen.
Es stellte sich am Ende der Diskussion heraus, daß beide Modelle interessante Ideen enthielten. Damit standen wir vor der Frage, welches der beiden Modelle wir in die Anforderungsdefinition eines fiktiven Systems übernehmen sollten. Da in beide Konzepte schon viel Arbeit und Schweiß investiert wurde, entschlossen wir uns, keines der Konzepte dem Projekt vorzuenthalten.

3.6.4 Das Ergebnis

3.6.4.1 Die Austauschformate

Neben denen am Anfang gemachten Erläuterungen enstand als Ergebnis eine Tabelle der Programme mit ihren Austauschformaten.

3.6.4.2 Die Ausgabe

Als Ergebnis der Ausgabe entstanden drei Ergebnisse:

- Das Idealbild
- Modell 1
- Modell 2

Die beiden Modelle hatten eines gemeinsam, sie bezogen sich beide nur auf die `WYSIWYG`-Sicht. Die anderen Sichten wie `ASCII` oder `EPS` ließen sich in normalen Textfenstern darstellen und bereiteten daher keine Probleme.

3.6.4.2.1 Das Idealbild Das Idealbild ging von einem Treiberkonzept für jedes Gerät aus, wobei ein Gerät ein Bildschirm, eine Grafikkarte oder ein Drucker sein konnte.
Das ideale Bild mußte diese Treiber weiterhin für unterschiedliche Rechnerarchitekturen und Benutzungsoberflächen realisieren.
Wichtig erschien uns in diesem Konzept, daß jedes Gerät auch optimal ausgenutzt werden sollte, eine Grafikkarte also über den evtl. eingebauten zusätzlichen Prozessor angesteuert wird oder ein 600-DPI-Drucker nicht mit 300 DPI druckt.

3.6.4.2.2 Modell 1 Der Aufbau der Grafik sollte unter Ausnutzung der Routinen des Betriebssystems geschehen. Jedes Betriebssystem und/oder die darauf aufsetzende Benutzungsoberfläche liefert schon eine Anzahl von Routinen, die die Darstellung von z.B. Texten in Grafikfenstern unterstützen. Das Problem liegt nun darin, daß die Routinen von System zu System völlig unterschiedlich sein können.
Daher sah Modell 1 vor, daß der Aufbau des WYSIWYG-Fensters in einem zentralen Modul vorgenommen werden sollte. Das zentrale Modul (Black Box) sollte genau definierte Schnittstellen besitzen, die für jedes System identisch sein sollten. Der interne Aufbau der Black Box wäre dagegen von System zu System unterschiedlich, da die Black Box die Routinen des Systems ausnutzen sollte.
Die Black Box sollte auf ein System aufsetzen. Da wir uns an den realen Gegebenheiten orientieren wollten, und die Realität an der Universität SUN hieß, schlugen wir als Benutzungsoberfläche XWindows vor. Für XWindows existiert eine Erweiterung, XView, die die Programmierung von XWindows erleichtert und viele Routinen bietet, die für die Black Box nützlich sein können.
Ein weiterer Pluspunkt der Lösung bestand darin, daß mittels des Unix-Derivates `Linux` eine Verfügbarkeit des Systems auch in der PC-Welt gewährleistet war.

Für das Drucken des Dokuments sollte dieses aus der internen Sicht des Dokuments nach Encapsulated Postscript (kurz EPS) gewandelt werden. EPS ist eine sogenannte Druckerbeschreibungsprache, ein PS-fähiger Drucker liest eine EPS-Datei und führt die Befehle aus. Postscript hat sich in den letzten Jahren zu einem weit verbreiteten Standard entwickelt, der systemübergreifend existiert. Damit war das Modell eine systemunabhängige Lösung, die Portierbarkeit der Textverarbeitung wäre für die Ausgabe sehr einfach.
Ein Effekt der Lösung war, daß die zusätzliche Sicht EPS existierte. Damit war das Exportieren des EPS-Formates automatisch möglich, es wurden dafür keine weiteren Übersetzer benötigt.
Das Hauptproblem war jedoch der Umstand, daß die Sichten Bildschirm und EPS unterschiedlich berechnet wurden. Es konnte im WYSIWYG-Fenster zu einer Darstellung des Dokumentes kommen, die nicht der des Druckbildes entspräche. Es wurden zwei Maschinen zur Berechnung der

Darstellung des Dokumentes verwendet, die ihre Daten evtl. aus unterschiedlichen Quellen beziehen sollten. So stammt der Zeichensatz zur Darstellung auf dem Bildschirm aus XWindows, der für den Drucker ist ein Postscriptzeichensatz. Jede der beiden Maschinen skaliert den Zeichensatz evtl. unterschiedlich, so daß z.B. ein zusätzlich im Dokument befindlicher Kreis im Vergleich zum Text auf dem Bildschirm kleiner wirkt als auf dem bedruckten Papier.

3.6.4.2.3 Modell 2 Die Bildschirmdarstellung in Modell 2 basiert auf der Visualisierung eines EPS-Seitenfiles durch einen entsprechenden Postscriptinterpreter und arbeitet seitenorientiert. Für die Darstellung einer Textseite in einem solchen System, muß nun der im folgenden kurz beschriebene Weg durchlaufen werden.
Am Anfang dieses Weges steht die Transformation der Textseite, welche in der Internen Struktur gehalten wird, in ihre äquivalente EPS-Notation mittels eines speziellen Übersetzers. D. h. jedes Textelement dieser Seite wird mit der vom Formatierer festgelegten Seitenposition und seinen Attributen (Font, Fontgröße, fett, kursiv etc.) in ein zur Visualisierung benötigtes EPS-Statement überführt. Eine so transformierte Seite wird in einem EPS-Seitenfile abgelegt und in einer FORAUS/EPS-Seitenverwaltungseinheit angemeldet. Diese reagiert mit der WYSIWYG-Darstellung der Seite am Bildschirm.
Änderungen am und im Text werden wie folgt im Modell behandelt. Die Art der Änderung, sowie deren aktuelle Bildschirmposition, werden an eine Beziehungseinheit gemeldet. Diese baut die zur Bearbeitung der Änderung benötigten Relationen zwischen Bildschirmposition, EPS-Seitenfile und der Internen Struktur auf. Danach wird die Änderungen in der Internen Struktur festgeschrieben. Anschließend wird der Prozeß zur Neuformatierung und Neuübersetzung mit anschließender Neudarstellung initiiert.
Der Vorteil dieses Modells liegt in dem verwendeten Ausgabeformat, welches sich sowohl zur direkten Ausgabe auf einem postscriptfähigen Medium (Drucker oder Bildschirm) eignet, als auch als genormtes Austauschformat fungieren kann. Ein weiterer Vorteil ist, daß für jede Klasse von Dokumenten immer eine WYSIWYG-Darstellung erzeugt wird, da Art und Umfang der Auszeichnungen keine Rolle spielen.
Eine entscheidende Schwäche des Modells liegt in dem hohen Verwaltungsaufwand für die entsprechende Darstellungen des Textes in EPS-Form, sowie in dem umständlichen Weg, auf dem Änderungen in der Internen Struktur bekannt gemacht werden.
Die Idee für ein solches Modell entstand in Anlehnung an die Funktionsweise eines quelltextfähigen Debuggers, der zwischen Quelltext und Quellcode eine eindeutige Beziehung herstellen kann.

3.6.5 Das Schlußwort

In der Gruppe wurde mehr erarbeitet, als eigentlich notwendig war. So haben wir drei Modelle und nicht nur eines entwickelt. Für uns hat sich das Entwickeln alternativer Modelle aber gelohnt, da durch die entstandenen Diskussionen viele Schwachpunkte entdeckt wurden und außerdem bei allen Gruppenmitgliedern ein großes Verständnis der Materie zu Ende der Arbeit vorlag.
Ein klein wenig bedauerlich fanden wir den Umstand, daß die Arbeit der Austauschformate letztendlich umsonst war, da diese im späteren Projekt nicht mehr berücksichtigt wurde.

Bei der Realisierung wurde später „ungefähr“ Modell 1 gewählt. Das Modell wurde nur ungefähr realisiert, da die Black Box nur für das System `XView` implementiert wurde.

3.7 Benutzerbefragung

Jan Hiller, Andreas Hinken

3.7.1 Motivation einer Benutzerbefragung

Beim Zusammenstellen der Funktionalitäten, die das zukünftige Textsystem ForauS bieten sollte, stellte sich uns die Frage, welche Eigenschaften Anwender einer Textverarbeitung am meisten nutzen bzw. von einer besseren Textverarbeitung erwarten. Da unser Blick durch die ständige Konfrontation mit dem Thema schon subjektiv getrübt war, entschlossen wir uns, die Anwender selbst nach ihren Vorschlägen zu befragen.

3.7.2 Entwicklung

Zunächst sollte eine Benutzerbefragung per E-Mail durchgeführt werden, um möglichst viele Anwender erreichen zu können. Aufgrund der schon vorangeschrittenen Zeit, der hauptsächlich universitären Verbreitung dieses Mediums und um einen repräsentativen Durchschnitt zu erreichen, haben wir uns für die Befragung mittels eines Fragebogens entschieden.
Dabei wurden zur Ermittlung des Benutzerprofils folgende Informationen verlangt:

- Um den Anwendungsbereich der Nutzer einschätzen zu können, haben wir die verwendete Systemplattform abgefragt. Diese sollte zusätzlich zur Beurteilung dienen, auf welche Systeme das fertige Produkt später portiert werden könnte.
- Der Name der Textverarbeitung sollte uns bei der Auswahl der Import-Filter für fremde Dateiformate behilflich sein.
- Um einen Eindruck über den Stellenwert der Nutzung zu erhalten, wollten wir wissen, wie häufig eine berufliche oder private Verwendung erfolgt.
- Schließlich ging es bei der letzten Frage des Grundinformationsblocks um die am häufigsten verarbeiteten Dokumentarten. Hier wurden u.a. Briefe und Texte vorgeschlagen.

Die am meisten verwendeten Funktionen der Textverarbeitung sind im „Funktionalitätsblock“ erfragt worden. Zusätzlich haben wir einige unserer darüberhinausgehenden Funktionen als neue Merkmale einer Textverabreitung vorgeschlagen und die Resonanz überprüft.
Jeder Projektteilnehmer wurde mit der Aufgabe betraut eine gewisse Zahl von Personen innerhalb eines fest definierten Zeitraumes zu befragen.

3.7.3 Ergebnis

Die Befragung zog sich zunächst länger hin, als ursprünglich eingeplant. Nachdem schließlich alle Umfragebögen zur Befragungsgruppe zurückgekehrt waren, konnte mit der Auswertung begonnen werden.
Schnell zeigte sich, daß die Befragten nur eine sehr beschränkte Auswahl der am Markt befindlichen Software-Produkte verwenden. Enttäuschend stellte sich für uns die weitere Auswertung der Antworten dar. Selten gingen die Antworten über die von uns exemplarisch angegebenen Vorschläge hinaus. Einige der Fragebögen wurden nicht mit der nötigen Sorgfalt ausgefüllt, so daß die Ergebnisse teilweise nicht verwertbar waren. Manche Bögen ließen darauf schließen, daß es den befragenden Personen nicht gelungen ist, die Inhalte und Absichten der Fragen hinreichend darzustellen.

Aufgrund dieser Erfahrung sollte im Anschluß an die Feststellung der Anforderungsdefinition eine weitere Befragung mit verändertem Fragenkatalog, unter Berücksichtigung der gesammelten Erfahrungen, durchgeführt werden. Diese sollte insbesondere die im Anforderungsdokument festgelegten Funktionalitäten mit den realen Benutzerwünschen abgleichen. Da wir mit der Festlegung der genauen Anforderungsdefinition unser Zeitlimit im Projektverlauf bereits überschritten hatten, wurde auf die zweite Umfrage verzichtet.

3.8 Software-Ergonomie

Volker Schmidtke

3.8.1 Motivation

Schon beim ersten Treffen der Projektgruppe wurde eines deutlich: Wir wollten ein grafikorientiertes, benutzungsfreundliches System zur Erstellung von Dokumenten entwickeln. Das Arbeiten mit einer logischen Dokumentenstruktur und die Technik des Multi-View (mehrere Sichten auf ein Dokument zur gleichen Zeit) sollten zum Einsatz kommen. Bei beiden Elementen spielt die Interaktion mit dem Benutzer eine gewichtige Rolle. So sollte erreicht werden, daß

- der Benutzer das System leicht bedienen kann (Erwartungskonformität),
- die Bedienung möglichst leicht erlernt werden kann (Erlernbarkeit),
- die Dialoge weitgehend konsistent für den Benutzer erscheinen (Konsistenz) und
- das Bearbeiten der logischen Dokumentenstruktur weitgehend frei von technisch bedingten Handlungsvorgaben sind (Transparenz).

Ein zentraler Aspekt von ForauS ist die Benutzungsoberfläche, sprich die Visualisierung dessen, was das System an Ein- bzw. Ausgaben dem Benutzer anbietet und ihm so eine Interaktion mit dem Computer ermöglicht. Die leichte Bedienbarkeit (aus der Sicht des Anwenders betrachtet!) der zu entwerfenden Benutzungsoberfläche zeichnet die Benutzungsfreundlichkeit aus. Die Gestaltung der Oberflächenelemente und die Gestaltung der Interaktion (Dialog mit dem Computer) ist Aufgabe der Entwickler.
Hierzu hatten uns die Projektbetreuer Michael und Mattias schon beim ersten Projekttreffen einige Anregungen gegeben, so daß wir uns ein vages Bild vom Zielsystem machen konnten. Die Meßlatte für unser Textverarbeitungssystem war dabei hoch angelegt![2]
Vorstellbar war zu diesem Zeitpunkt ein System mit Funktionen wie

- automatische Inhaltsverzeichnisgenerierung,
- automatische Kapitelnumerierung,
- automatische Seitennumerierung,
- automatischer Zeilenumbruch,
- automatische Querverweise und Bibliographieverwaltung,

[2] Die beiden wissenschaftlichen Mitarbeiter präsentierten uns am 22.10.92 „Die Wahrheit über ForauS“, worin ihre Vorstellungen zusammengefaßt wurden.

- automatischer Seitenumbruch,
- mehrere Schrifttypen, -schnitte, und -größen,
- automatische Index- und Glossargenerierung,
- automatische Fußnotenverwaltung,
- verschiedene Absatzformate,
- Grafikintegration,
- Tabellenintegration,
- automatische Generierung von Abbildungsverzeichnissen,
- Unterstützung von mehreren Sichten auf das gleiche Dokument.

Der Benutzer sollte im WYSIWYG-Modus edieren und den Text gestalten können, dabei jedoch so wenig wie möglich mit satztechnischen Problemen zu tun haben.[3] Als Beispiel für einen „echten“ Multi-View wurde das Programm „IslandPaint/IslandPresents“ [Isl92] vorgestellt, welches dieses Feature sehr anschaulich verdeutlicht.[4]
Gerade die Besonderheiten des angedachten ForauS-Systems motivierten uns im ersten Semester sehr. Vermutlich war auch der Ehrgeiz, eine Textverarbeitung zu entwickeln, die neuartig für uns war, die treibende Kraft. Auf alle Fälle sollte uns die Arbeit interessieren, viel Spaß machen, fordern und nicht überfordern, denn die Gestaltung und der Umfang des Systems konnte weitgehend von uns selbst bestimmt werden.

3.8.2 Das erste Projektsemester

Diese aufgestellte Funktionsfülle des ForauS-Systems wirft eine Menge Fragen zur Software-Ergonomie auf:

- Wie ist die Aufgabenverteilung von Benutzer und Computer bzw. welche Aufgaben bleiben für den Benutzer?
- Worauf kommt es bei Visualisierung, insbesondere bei Multi-View-Darstellungen an?
- Wie sollten welche Dinge visualisiert werden?
- Welche Funktionalitäten sollten durch den Benutzer beeinflußbar sein?

Da diesen Fragestellungen nachzugehen war, mußten wir uns zunächst eine theoretische Basis schaffen, um die Anforderungen an unsere zukünftige Textverarbeitung näher zu definieren.
Schon früh erkannten wir, daß dies im Wesentlichen durch die Abarbeitung der folgenden Punkte geschehen könnte:

- Vergleich bestehender Systeme,

[3] Hiermit ist nicht die Grammatik, sondern das Aussehen des Textes auf dem Papier/Bildschirm gemeint.
[4] Multi-View ist nach unserer Definition eine Darstellungstechnik, bei der gleichzeitig unterschiedliche Sichten auf dieselbe Informationsmenge dem Benutzer angezeigt werden können. Als Erweiterung zur Mehrfenstertechnik dürfen diese Sichten verschiedene Präsentationen (reduzierte, transformierte und exponierte Darstellungsformen) mit zum Teil speziellen Manipulationsmöglichkeiten beinhalten.

- Vergleich verschiedener Oberflächenstandards,
- Betrachtung der theoretische Grundlagen aus der Software-Ergonomie,
- eigene Überlegungen / Planung der Funktionalität unseres Systems,
- gezielte Benutzerbefragung.

Es bildeten sich zu diesem Zweck mehrere Arbeitsgruppen, die zu den verschiedenen Themenrichtungen Informationen sammelten und anschließend dem Projektplenum präsentierten. Dies geschah ganz unabhängig davon, ob nun tatsächlich die eine oder andere Funktionalität in unserem System realisiert werden konnte bzw. sollte. Diese Projektphase war von uns deshalb auch besonders kreativ bearbeitet worden.
In den folgenden Textabschnitten soll auf die Entstehung der Anforderungsdefinition eingegangen werden. Die Ideen, Konzepte, die Vor- und Nachteile von Vorgehensweisen im Projekt, sowie die letztendlichen Entscheidungen, die zu der Anforderungsdefinition und dem Programm geführt haben, möchten wir damit beschreiben. Das Ergebnis selbst liegt der Dokumentation anbei.

3.8.2.1 Vergleich bestehender Textsysteme

Wie schon oben angesprochen, besteht eine Textverarbeitung für den Anwender aus einer Sammlung verschiedener Funktionalitäten, die einen Text nach seinen Wünschen manipulieren können. Auf dem Projektseminar am 18. und 19.11.92 in Syke wollten wir deshalb herausarbeiten, welche Akzente die gerade auf dem Markt befindlichen Produkte setzen, welche Funktionen sie dem Benutzer anbieten, und wie unser System in diesen Produktbereich passen könnte.
Hierzu verglichen wir gleich mehrere Textverarbeitungsprogramme miteinander, die auf den Atari ST-, Apple MacIntosh- bzw. PC-Rechnern zu sehen waren.

- Microsoft WinWord (PC)
- Pagemaker (PC)
- FrameMaker (Apple MacIntosh)
- Calamus (Atari ST)
- Tempus Word (Atari ST)

Wir hörten uns dazu die Erfahrungsberichte an und diskutierten über die Oberflächen der vorgestellten Programme. Die Schwierigkeit bei diesem Vergleich war natürlich, daß wir kein System hatten, das auch nur annähernd über Multi-View (nach unseren Ansprüchen gesehen) oder eine logischen Strukturierung des Dokumentes verfügten. Dennoch konnten wir bei den untersuchten Programmen bestimmte Eigenschaften festmachen.

- Die Grundfunktionen der Textverarbeitungssysteme sind bei den meisten Programmen identisch. Sie basieren hauptsächlich auf einer physikalischen Auszeichnung von einzelnen Buchstaben, Worten oder Absätzen. Eine Strukturierung der Dokumente erfolgt nicht bzw. nur ansatzweise durch das Erstellen von Elementgruppen mit gleichen physikalischen Attributen (siehe Microsoft WinWord).
- Viele Programme (insbesondere das Programm „Calamus“) bieten die Möglichkeit der freien Bearbeitung von Texten und sind daher eher dem Bereich des Desktop-Publishing (DTP) zuzuordnen.

- Die meisten Programme bieten den Anwendern eine kaum zu bedienende Funktionsvielfalt. Viele Funktionen werden von den Anwendern nur äußerst selten verwendet.
- Obwohl die Programme sich in der Oberflächendarstellung bis auf wenige Ausnahmen nicht sehr unterscheiden, gibt es starke Unterschiede beim Handlungsablauf.
- Hilfestellungen werden mal mehr — mal weniger angeboten und sind im Allgemeinen von schlechter Qualität.
- Die vom Benutzer zu beherrschende Dialogkomplexität ist gerade bei größeren Systemen (z.B. WinWord) nur schwer zu durchdringen. Vieles ist nicht, wie oftmals versprochen, selbsterklärend.
- Im Allgemeinen erkennt man einen Trend der Programme zu immer mehr Funktionen (z.B. eingebauter Formelsatz, Tabellensatz, Präsentationsgrafiken, Tabellenkalkulation, Grafikprogramme). Man kann beobachten, daß je schneller die Rechnerklasse ist, auf der die Systeme laufen, desto umfangreicher sind die Möglichkeiten zur Manipulation von Dokumenten.

Je nach System muß der Benutzer also mehr oder weniger stark gestalterisch bei der Erstellung des Dokumentes mitwirken, da eine vollständig automatisierte Erzeugung von Dokumenten weder momentan technisch möglich noch besonders angenehm für den Benutzer sein kann (aus softwareergonomischer Sicht).
Der Unterschied zwischen Anspruch des Anwenders und die Wirklichkeit beim Ausdruck des Textes hängt somit von den Fähigkeiten des Anwenders selbst als auch von Fähigkeiten des Programmes ab.
Dies kostet dem Benutzer natürlich zusätzliche Arbeit, die er sicherlich besser mit inhaltlichen Aspekten des Textes verbringen könnte. Um dabei eine gute Satz- bzw. Layoutqualität zu erreichen, ist es außerdem zwingend notwendig, daß der Benutzer über Fachwissen verfügt, was bei den meisten Anwendern nicht angenommen werden kann. Es ist also kein Wunder, warum — gerade bei der Benutzung von Textverarbeitungen mit hohem DTP-Charakter — eher schlechte Dokumente entstehen.[5]
Ein Lösungsansatz zu diesem Problem ist, daß man das Erstellen eines Dokumentes für den Benutzer strukturierter und die Formatierung und Eingabe des Textes automatisierter geschehen läßt. Dieses Vorhaben kann natürlich nur mit besonders gut strukturierbaren und automatisierbaren Dokumentarten geschehen, wie z.B. bei wissenschaftlichen Texten, Protokollen, Standardbriefen etc.. Dokumente, bei denen es auf spezielle Typographie und auf spezielle Layout-Gestaltung ankommt (zum Beispiel Werbedrucke, Zeitungen, Geburtstagskarten usw.) sind nur sehr schwer zu automatisieren. Sie können deshalb nicht das Ziel unseres Projekts sein.
Grundsätzlich kann man die Strukturierung von Dokumenten auf mehrere Arten realisieren: Die Beschreibung der physikalischen Struktur - und die Beschreibung der logischen Struktur eines Dokumentes.
Die physikalische Auszeichnung[6] von Text (wie zum Beispiel bei WinWord angewandt) mittels

[5] Aus eigener Erfahrung sind solche Programme zur Erstellung von Vereinszeitungen und Werbeschriften gut zu gebrauchen. Für größere Texte, die der Leser entspannend/ermüdungsfrei lesen möchte, sollte man stärker auf die einheitliche Form des Textes Wert legen. Die Befolgung typographischer Regel ist dazu unerläßlich.

[6] Eine physikalische Auszeichnung bestimmt das Erscheinungsbild (Größe, Zeichensatz, Schnitt etc.) der einzelnen Zeichen und Textabschnitte sowie die Formatierung von Absätzen (rechtsbündig, zentriert, linksbündig, Blocksatz, Einrückung, Silbentrennung etc.) relativ zu einem bestimmten Seitenlayout. Korrekterweise bedeutet die „physikalische Auszeichnung" die Auszeichnung der physikalischen Struktur eines Dokumentes, denn die Eingabe des Textes und die Bestimmung der Formatierung geschieht sowieso auf physikalischer Ebene (sie ist eigentlich die Bestimmung der Schwärzung auf dem Papier).

vorgefertigter Formatierungs-Makros[7] und sogenannten Dokumentenvorlagen läßt den Benutzer fast alle Freiheiten der Gestaltung seines Dokumentes. In Unkenntnis typographischer Regeln (uns Studenten eingeschlossen) werden dabei oft sehr viele (meist zu viele) Auszeichnungen verwendet, was zu mehr oder weniger schönen Dokumenten führen kann, deren Strukturierung oft nicht mehr stimmig[8] ist und somit beim Leser für Irritationen sorgt.[9]
Für die Benutzungsoberfläche bedeutet die physikalische Auszeichnung, daß sehr viele Knöpfe und Schalter in vielen Dialogmasken für die Steuerung der verschiedenen Attribute und Formate eingesetzt werden müssen, was natürlich die Übersichtlichkeit negativ beeinflußt. Diese Eigenschaft ist besonders bei Tempus Word und WinWord zu beobachten. Mit steigender Komplexität der Gestaltungsmöglichkeiten steigt daher auch die Komplexität der Benutzungsoberfläche.
Bei der logischen Beschreibung[10] von Dokumenten ist die Komplexität der Textgestaltung um ein Vielfaches geringer als bei der physikalischen Auszeichnung. Dies liegt daran, daß nicht mehr das Layout im Vordergrund steht, sondern der Text und die Dokumentenstruktur. Das Wissen über die Repräsentation der logischen Elemente bleibt im Wesentlichen dem Anwender verborgen, da die Layout-Regeln von Fachleuten zentral für (fast) alle strukturierten Dokumentklassen beschrieben werden können. Dementsprechend einfach fällt die Oberfläche des Textverarbeitungssystems aus, was die Bedienung zum Teil sehr erleichtern kann. Eine Schwierigkeit ist allerdings, daß diese Art von Textverarbeitungssystemen noch nicht in nennenswerter Zahl auf dem Markt präsent sind, und wir somit gewissermaßen Neuland betreten, ohne auf wertvolle Erfahrungen zurückgreifen zu können.

3.8.2.2 Erarbeitung der Anforderungen des ForauS-Systems

Die Umsetzung der Ideen einer strukturierten Textverarbeitung mit Multi-View in eine Benutzungsoberfläche und eine dazu passende Handhabung der Oberfläche sollte die am 19.11.92 ins Leben gerufene Benutzungsoberflächen-Gruppe ausarbeiten. Die Gruppe bestand aus vier Mitgliedern: Lu Lei, Van Son Le, Rainer Schmidtke und Volker Schmidtke.
Im Brainstorming-Verfahren analysierten wir zuerst, was die Schwerpunkte unserer Arbeit sein sollten. Da zu diesem Zeitpunkt noch keine konkreten Funktionen unserer zukünftigen Textverarbeitung feststanden (außer groben Skizzen und vagen Vorschlägen), konzentrierten wir uns zunächst auf allgemeinere Themen wie

- die software-ergonomischen Grundlagen der Wahrnehmung und Informationsverarbeitung und
- die Richtlinien für eine ergonomische Dialoggestaltung unter Verwendung von Normen und Oberflächenstandards.

[7] Die Formatierungs-Makros sind Kommandokürzel, die auf einem Textabschnitt gleich mehrere Zeichen-/Absatz-Attributierungen aktivieren. Der Textabschnitt ist vom Benutzer frei definierbar.

[8] Wenn man z.B. beim Kenntlichmachen von wichtigen Begriffen unterschiedliche Formatierungen/Attribute benutzt. Aus eigener Erfahrung passiert dies recht häufig!

[9] Dies hängt sicherlich damit zusammen, daß Menschen eine Bedeutung in der Formatierung eines Textes sehen. Siehe auch Kapitel „Software-ergonomische Grundlagen und Richtlinien“, wobei die Bedeutung der Formgebung analog zur Textformatierung zu sehen ist.

[10] Die „logische Beschreibung“ eines Dokumentes repräsentiert dessen logische Struktur, nicht das Aussehen. Statt jedem Textbereich eine Anzahl von Formatierungsattributen zuzuordnen, werden die Formatierungs-Regeln nur einmal für jedes logische Element (z.B. Titel, Überschrift, Absatz etc.) außerhalb des Dokumentes definiert. Gleiche Textelemente können dabei mit unterschiedlichen Regeln auf einer Seite formatiert werden, wenn sie einen anderen logischen Kontext (und damit eine Ordnung) besitzen. Wichtig gegenüber der Auszeichnung der physikalischen Struktur ist, daß die sogenannte „Weißraumverteilung“ völlig vernachlässigt werden kann, da diese Informationen hierbei nicht speziell zu einem Dokument gehören.

Die software-ergonomischen Grundlagen sollten bei uns die theoretische Basis bilden und - für diejenigen, die sich mit der Software-Ergonomie noch nicht auseinandergesetzt haben - gleichzeitig einen Einstieg in diesen Themenbereich bieten. Die physischen und psychischen Eigenschaften des Menschen, die Softwareentwickler zu berücksichtigen haben, stehen dabei im Vordergrund.
Normen wie die DIN 66 234 [DIN88] und ISO 9241 [DIN94] helfen dabei, die software-ergonomischen Grundlagen in den groben Entwurf der Benutzungsoberfläche eines Programms umzusetzen. Sie gehen deshalb ein in die Verfassung von bestimmten Anforderungskriterien beim Entwurf und der Implementierung.
Letztlich gibt es noch die Oberflächenstandards, die auf den jeweiligen Computer- und Betriebssystemen die Präsentation der Elemente von (graphischen) Benutzungsoberflächen festschreiben. Von Interesse waren für uns besonders diejenigen Standards, die auf Systemen laufen, auf denen wir wahrscheinlich implementieren würden.
In Frage kamen entweder die Richtlinien des Apple Macintosh - Rechners (Apple-Style-Guideline [App87] oder die der SUN-Computer mit einer X-Window-Schnittstelle (also OSF-Motif- bzw. Open-Look-Aufsatz). Sie sind ein wichtiger Hinweis bei der Ausgestaltung der Dialoge des Systems und zeigen eine praktikable Umsetzung (wie Cut/Copy/Paste, Drag and Drop, Auswählen und Anzeigen von Objekten) von Programmfunktionen auf einem bestimmten System.
Mit dem weiteren Fortschreiten des Semesters konnten wir auch speziellere Themen, wie die des Mehrbenutzerbetriebs und einige (Standard-) Funktionen der „Advanced Features-“ und „Standardfunktionen-“ Arbeitsgruppe aufgreifen (z.B. Versionsverwaltung, Undo/Redo etc.). So konnten wir mit jedem internen Gruppentreffen die Ergebnisse der anderen Gruppen, die in den Projektplenen zusammengetragen wurden, in unsere Überlegungen mit einbauen.
Das Resultat dieser thematischen Aufarbeitung sollte ein sogenannter „Kriterienkatalog“ sein, in dem beschrieben wird, welche ergonomischen Anforderungen an das System zu stellen sind. Die Idee zur Strukturierung des Kriterienkatalogs stammt dabei in den Grundzügen aus [GJ89].
In den folgenden Wochen beschäftigten wir uns in der Teilgruppe mit dem Sichten und Aussuchen von Literatur zu den Themen „Open Look Style Guideline“ [Sun90], „Apple Style Guideline“ [App87], „Grundlagen der Software-Ergonomie“ und „Richtlinien zur ergonomischen Dialoggestaltung“. Wir koordinierten die Begriffe und Prioritäten der Informationen und halfen uns gegenseitig, die gefundenen Resultate und Schwerpunkte zu bewerten. Während die Literatur zu den Guidelines im Wesentlichen feststand, mußten für die software-ergonomischen Grundlagen zahlreiche Artikel gelesen werden. Insbesondere zu erwähnen wären an dieser Stelle die Artikel [KTR84] und [Opp89], die uns das benötigte Hintergrundwissen lieferten.

3.8.2.3 Software-ergonomische Grundlagen und Richtlinien

Kernpunkte des Vortrags am 29.01.93 waren die software-ergonomischen Grundlagen. Hierzu zählen die Gesetze der Wahrnehmung, die Art und Weise wie Menschen mit Maschinen kommunizieren können (falls man hier überhaupt von Kommunikation reden kann!) und die Funktionsweise des Wissenerwerbs beim Menschen. Aber auch Modelle der Mensch-Maschine-Kommunikation und das IFIP-Benutzungsschnittstellen-Modell[11] wurden erläutert. Gestaltungshilfen bei der Einhaltung dieser Grundlagen sind die Richtlinien der DIN 66 234 und ISO 9241, die anschließend kurz skizziert wurden. Insgesamt stießen diese beiden Hauptthemen auf ein großes Interesse im Plenum. Der schon bekannte Stoff aus dem Grundstudium konnte also nochmals aufgefrischt bzw. ergänzt werden.

[11] Die Abkürzung IFIP steht für „International Federation of Information Processing“, eine Institution, die ein anwendungsunabhängiges Modell von Schnittstellen zwischen dem Benutzer, dem Rechner und der Arbeitswelt aufgestellt hat.

Aus den Grundlagen und den Richtlinien sowie weitergehender Literatur konnte ein Kriterienkatalog für eine fiktive Textverarbeitung entwickelt und ebenfalls dem Plenum vorgestellt werden. Dieser Kriterienkatalog umfaßte gleich mehrere Seiten ausführlichen Textes und konnte als Ausgangsbasis für die Anforderungsdefinition unseres fiktiven Systems dienen.
Ergänzt wurden die Ergebnisse durch andere Gruppen wie Standardfunktionalität, Advanced Features und Benutzerbefragung, die zuvor - aber auch noch im nachhinein in den Kriterienkatalog aufgenommen wurden.

3.8.2.4 Vergleich verschiedener Oberflächenstandards

Besonders die „Open Look Style Guideline“ [Sun90] war hierbei für uns von besonderem Interesse, schließlich stand zu diesem Zeitpunkt schon mehr oder weniger fest, daß wir nach diesem grafischen Standard unsere Oberfläche ausrichten wollten.[12]
Der Grund hierfür lag im Wesentlichen an der neuen und weiterhin im Ausbau befindlichen Rechnerausstattung der Universität, die sich fast ausschließlich mit der Planung eines Rechnernetzes bestehend aus SUN-Rechnern[13] befaßte. Nicht wenige Projektteilnehmer hatten sogar selbst Erfahrungen mit der Anwendung und Installation von „Open Look“ unter Linux[14].
Wir konnten also von einem auf X-Window basierenden Betriebssystem ausgehen, was natürlich die Wahl der benutzbaren Oberflächenstandards stark eingeschränkt hat.
Uns ging es allerdings im Vortrag nicht unbedingt um eine bestimmte Benutzungsoberfläche, sondern eher um die allgemeineren Konzepte, die hinter diesen X-Window-Oberflächen stehen. „Open Look“ sei daher exemplarisch als Vertreter dieser Gattung zu betrachten.
Die grafischen Elemente und deren Funktionen/funktionale Bedeutung, aber auch die zu Grunde liegenden Konzepte von „Open Look“[15] wurden von uns erarbeitet und in Übersichtsform vorgetragen. Für den Benutzer stellt sich das System als meist schlichte, auf Grafik basierende Oberfläche dar, die durch spezielle Eingabeelemente („Knöpfe“, „Schieberegler“, „Texteingabefelder“ u.ä.) benutzt wird. Formen und Anordnungen dieser Grafikelemente sind über Applikationen hinweg standardisiert, so daß sich der Benutzer nicht in jeder Anwendung neu orientieren muß. Insbesondere der Austausch von Daten zwischen mehreren Applikationen ist gut geregelt.[16] Im Mittelpunkt der Oberfläche steht jedoch nicht die Grafik (im Sinne von Bildern, Piktogrammen etc.), sondern eine möglichst einfache und schnelle Bedienung der Applikationen über sogenannte „Hot-Keys“, Tastaturkombinationen und Maustasten.
Für die spätere Implementierung kam uns das hierbei gewonnene Wissen auch sehr entgegen. Als Kontrast zu diesem Konzept haben wir Apples Style-Guidelines gesetzt, wobei es uns besonders auf die Darstellung der Unterschiede zu „Open Look“ ankam. Hauptunterschiede sind die größere Einheitlichkeit auch über verschiedene Applikationen hinweg, die Metapherbildung mit direkter Manipulation von Objekten und der gezielte Einsatz von Bildern und Farben. Diese Eigenschaften ermöglichen den Benutzer ein ganzheitliches Umgehen mit dem Computer, sie sind jedoch nicht notwendigerweise effizienter.

[12] Auf Anraten der wissenschaftlichen Mitarbeiter, die die XView-Bibliotheken (und damit „Open Look“) als besonders einfach beschrieben! Auch die mögliche Anwendung der XView-Bibliothek für ASpecT (eine funktionale Programmiersprache, entwickelt an der Universität Bremen und eine der mögliche Programmiersprachen für die Implementierung) war ein Entscheidungsgrund hierfür. Als wirkliche Alternative hätten wir OSF-Motif verwenden können. Diese Idee wurde allerdings nicht weiter verfolgt.

[13] Die darauf laufenden Betriebssysteme SOLARIS bzw. BSD-Unix bieten die Option einer X-basierten graphischen Oberfläche.

[14] Linux ist eine frei verfügbare UNIX-Umsetzung.

[15] Diese sind insbesondere geprägt durch die Begriffe „Einfachheit“, „Konsistenz“ und „Effizienz“.

[16] Gerade bei der Übertragung von Text (-Teilen).

Leider konnten wir diese Erkenntnisse bei der späteren Implementierung weniger verwenden, da wir sonst den „Open Look“ - Standard verlassen hätten. Dennoch war dieser thematische Ausflug für viele Projektteilnehmer sicherlich ein Bereicherung.

3.8.2.5 Die schriftliche Ausarbeitung

Die schriftliche Ausarbeitung bestand in unserer Gruppe einerseits aus der ausführlichen Beschreibung des Inhalts des Vortrags und andererseits aus der Komplettierung des Kriterienkatalogs. Der ursprünglich geplante Abgabetermin bis Mitte/Ende Januar konnte aufgrund der Verschiebung der Präsentationstermine und aufgrund von Koordinationsschwierigkeiten in der Gruppe, die infolge der vorlesungsfreien Zeit auftraten, nicht eingehalten werden. Trotz des großen Umfanges der Arbeit schafften wir die Bearbeitung bis zum Anfang des nächsten Semesters.
Zusätzlich zum mündlichen Vortrag haben wir versucht, konkrete Anforderungen zu defininieren. Sie beschreiben, wie unser zukünftiges System gestaltet werden soll und sind in tabellarischer Form zusammengetragen. Mit den „abschließenden Betrachtungen“ gingen wir zudem auch auf ein mögliches Konzept für die weitere Arbeit (bei Entwurf und Implementierung) ein.
Dieser Konzeptvorschlag bestand aus den Punkten

1. Überprüfung der Entwicklungsziele (auch durch den Anwender).
2. Eine mögliche Weiterentwicklung in Richtung Entwurf und Systemspezifikation.
3. Einsatz von Konzepten und Hilfsmitteln für die Implementierung und Portierbarkeit des Systems.

Leider fanden diese Punkte nur wenig Beachtung im Plenum und wurden aus zeitlichen Gründen, aber auch weil aktuellere Themen im Vordergrund standen, weitestgehend verdrängt.
Nachzulesen ist diese Arbeit in der ForauS-Dokumentation, Teil „Anforderungen an ein innovatives Dokumentenverarbeitungssystem“, Kapitel 6 und Anhang C.
Damit war das erste Semester abgeschlossen und wir konnten dazu übergehen, die erlangten Ergebnisse zu prüfen und ihnen Prioritäten zu vergeben.

3.8.3 Das zweite Projektsemester

Ziel des zweiten Semesters war die Vervollständigung der bisher gefundenen, eher vagen Anforderungen und das Anfertigen der daraus resultierenden endgültigen Systemanforderungen. Ein aus den fertigen Systemanforderungen abgeleiteter Systementwurf war zwar zu diesem Zeitpunkt noch nicht direkt vom Projekt geplant worden, rückblickend betrachtet wäre dies allerdings durchaus wünschenswert gewesen.

3.8.3.1 Konsequenzen aus der Benutzerbefragung für das ForauS-System

Die im ersten Semester gestartete Benutzerbefragung ist für die Oberflächengruppe, aber auch für die anderen Gruppen, von der Idee her sehr wichtig gewesen. Sie kann und sollte (nach software-technischen Erkenntnissen) zur Bestimmung der Anforderungen an das ForauS-System eingesetzt werden. Die Ergebnisse können uns helfen, ein System zu entwickeln, das auf die Wünsche und Bedürfnisse der möglichen Benutzer eingeht. Der Hauptwunsch der Befragten war die WYSIWYG-Sicht, was dem Stand der Technik entspricht und somit nicht fehlen sollte.
Ein weiterer mit unseren Vorstellungen übereinstimmender Punkt war die einfache Bedienung des Systems. Hier wurde jedoch von der Benutzerbefragungs-Gruppe versäumt, gezielter bei den

Befragten nachzuforschen, welche Kriterien sie denn nun an eine „einfache“ Benutzung stellen (überwiegend Tastatur- oder Maus-Benutzung, Art der Anzeigen und Dialoge am Bildschirm, Form der Hilfe und der Steuerungsmöglichkeiten), so daß wir uns selber Gedanken zu diesem Aspekt machen mußten. Offenbar sind aber die Benutzungsoberflächen der vorhandenen Textsysteme für viele Benutzer nicht mehr einfach zu bedienen, da sonst der Wunsch nicht geäußert worden wäre. Dies zeigt deutlich, daß die Funktionskomplexität in unserem ForauS-System für den Benutzer geringer und transparenter gestaltet werden mußte. Mit welchen Mitteln das geschehen kann, war uns zu diesem Zeitpunkt aber noch nicht ganz klar. Außer Frage stand jedoch die Konsequenz, daß die Gestaltung des Layouts und die Formatierung automatisierter realisiert werden mußte.
Der Wunsch nach „freier Positionierung von Grafik und Formel“ steht der Automatisierung jedoch zum Teil entgegen und kann nur durch einen Kompromiß gelöst werden.[17]
Um die aufgeführten Informationsmängel auszubessern, war eine weitere Befragung vorgesehen, die jedoch nicht in einem auswertbaren Zustand mündete.

3.8.3.2 Von den fiktiven zu den realen Anforderungen

Nachdem nun alle Arbeitsgruppen bis zum Anfang des zweiten Semesters ihre bisherigen Ergebnisse vorgestellt hatten (Advanced Features und Austauschformate, Software-Ergonomie, Typographie, Anwenderumfrage und Standardfunktionalität), mußten wir erkennen, daß die so gesammelten Informationen nicht konkret genug waren, um mit ihnen unser System innerhalb eines software-technischen Entwurfes zu beschreiben. Diese Lücke konnte auch die für alle relativ spät herausgegebene schriftliche Ausführung der „Anforderung an ein innovatives Dokumentenverarbeitungssystem“ [For93] nicht schließen. Statt jedoch sofort eine Konkretisierung der Anforderungen anzustreben, diskutierten wir zunächst über einzusetzende Programmiersprachen und weitere Problemsituationen, die im Zusammenhang mit den „fiktiven“ Anforderungen standen. Themen wie „Formulare“, „Datenschutz“, „Checkpoints“[18], sowie „Undo/Redo-“ und „Lock/Unlock-“ Funktionen beim Multiuser-Betrieb rückten plötzlich in den Vordergrund. Bestandteil davon waren auch software-ergonomische Überlegungen, die jedoch nur wenig fundiert beschrieben wurden. Für das Verständnis um das Zusammenspiel einzelner Systemkomponenten waren diese Themen von größter Bedeutung. Der größte Teil dieser Ideen wurde jedoch im späteren Systementwurf konzeptionell nicht verwirklicht, beziehungsweise als weniger wichtig eingestuft.
Es stellte sich heraus, daß wir uns für die Erstellung konkreter Systemanforderungen noch spezielleres Wissen in den Bereichen der „Realisierung einer logischen Strukturierung“ und der „Realisierung von typographischen Regeln“ aneignen mußten. Zu diesem Zweck bildeten wir neue Gruppen, die „SGML / Interne Struktur-Gruppe“ und die Gruppe „Typographie“ , was natürlich zur Folge hatte, daß die alten Gruppen entweder aufgelöst wurden oder nur noch am Rande weiterexistierten. Die Gruppe „Software-Ergonomie und Benutzungsoberfläche“ diskutierte daher nur noch sporadisch über die vorangegangenen Themen, da die meisten Gruppenmitglieder in die „SGML/Interne Struktur“-Gruppe wechselten und mit den neu gestellten Aufgaben vollauf beschäftigt waren. So fehlte uns auch die Zeit, um den interessanten Vorschlag eines graphischen DTD-Editor weiter zu verfolgen.
Erst nach den Präsentationen zur Typographie und internen Struktur, konzentrierten wir uns wieder auf die Vorbereitung der Implementierung der Oberfläche, obwohl die Ausformulierung der

[17] Dies könnte zum Beispiel durch eine Vorschlags-Formatierung geschehen, die der Benutzer nachträglich beeinflussen kann. Leider konnte diese Idee nicht im endgültigen Entwurf (siehe drittes Semester) durchgesetzt werden.

[18] Checkpoints stellen zeitliche Markierungspunkte dar, die sowohl als Handlungsabfolge (Historienübersicht), für wiederholte Ausführungen von Operationen als auch zur Rücknahme von Eingaben eingesetzt werden können.

internen Struktur weiterging. Da noch keiner aus unserer Gruppe Programmiererfahrungen unter XView/„Open Look“ hatte, suchten wir zunächst nach Informationen und Hilfsmitteln für den Entwurf und der späteren Implementierung. Unser selbstgesetzter Plan zur Vorgehensweise sah vor, daß zunächst die X11/XView - Schnittstelle untersucht wird, dann Leistungsanforderungen entwickelt, Module gebildet und die Schnittstellen dazu definiert werden, bis so nach und nach der fertige Entwurf feststeht. Eine - über die Applikation hinweg - einheitliche Gestaltung der Hilfen und Dialoge war von uns bis dahin angedacht, aber aufgrund der geringen Vergleichsmöglichkeiten und Erfahrungen noch nicht festgelegt worden.[19]
Bei der Sondierung möglicher Hilfsmittel zur Erstellung der Benutzungsoberfläche studierten wir mehrere Zeitschriften und Bücher, worauf sich einige Oberflächengeneratoren (englisch: „Graphical User Interface Manager“, kurz: GUIM) wie die Generierungssprache „TNT“ und andere als theoretisch sehr nützlich herausgestellt haben. Leider konnten wir diese Ideen bei der Implementierung nicht weiter verfolgen, da diese Werkzeuge entweder nicht zur Verfügung standen oder sie nicht problemlos an unser späteres Konzept adaptierbar waren.

3.8.3.3 Erste Überlegungen für den Entwurf

Mit den bisherigen schriftlichen Unterlagen war es uns nicht möglich, einen Entwurf festzulegen. Sie enthielten (meist nur) bloße Informationssammlungen dessen, was thematisch zum Themengebiet einer Textverarbeitung gehört. Also mußten wir versuchen, die Informationen als Funktionalitäten zu interpretieren und die dabei gewonnene konkretere Funktionalität unseres Systems soweit zu ordnen und zu gewichten, daß dazu ein modularer Entwurf entwickelt werden konnte. Dazu war es notwendig, erst einmal die bisherigen „fiktiven“ Anforderungen in „reale“ Anforderungen umzumünzen und die eher utopischen Vorstellungen dafür zu streichen. Dazu wollten wir auch inhaltlich abspecken, denn die Funktionsvielfalt hätte die Projektarbeit bei weitem zeitlich gesprengt. Im Gegensatz zu anderen Gruppen war unser Kriterienkatalog vom ersten Semester mit nur wenigen Korrekturen schon ausreichend für die Systemspezifikationen. Das eigentliche Festlegen der Schwerpunkte und die schriftliche Zusammenstellung geschah jedoch erst am Anfang des dritten Semesters. Die vorlesungsfreie Zeit nutzten wir zur Erkundung der C- bzw. ASpecT-Schnittstellen mit den entsprechenden Funktionsbibliotheken und den ersten Gehversuchen, Dialoge und Menüs unter „Open Look“ probeweise zu erstellen.

[19] Eine Orientierung bot uns das Programm „Framemaker“, welches wir näher betrachteten und das ebenfalls unter dem „Open Look“ - Standard lief. Bei der Auswertung der Dialogmasken konnten wir einige gute Ideen sammeln.

4. Implementierung

4.1 Softwareentwurf

Kai Siegele

Der Softwareentwurf sah eine Aufteilung des gesamten Programmes in mehrere unabhängige Module vor. Das Problem, die Verarbeitung eines Textes, wurde in mehrere Teilprobleme zerlegt. Jedes Modul sollte eines der Teilprobleme lösen. Insgesamt waren fünf Module vorgesehen. Das Modul der internen Struktur ließ sich in drei Untermodule unterteilen. In einem Untermodul wird der Text, sowie die logischen Elemente, aus denen der Text besteht, verwaltet. Das zweite Untermodul speichert die zugehörige DTD und gibt Auskunft, ob die logischen Elemente anhand der DTD korrekt angeordnet sind. Die mit den logischen Elementen verbundenen „Presentation Rules“ werden in einem weiteren Untermodul gespeichert. Das Modul des Parsers sorgt dafür, daß am Anfang der Verarbeitung die DTD, der Text und die „Presentation Rules“ geladen, auf syntaktische Korrektheit überprüft und durch entsprechende Funktionsaufrufe in der Internen Struktur aufgebaut werden. Das Formatierermodul erzeugt aus den Angaben, die in der Internen Struktur gespeichert sind, eine Zwischendarstellung, die Zusatzinformationen für die Formatierung des Textes enthält. Durch Aufruf der Funktionen des Modules Ausgabe wird aus der Zwischendarstellung die Ausgabe auf dem Drucker bzw. dem Bildschirm erzeugt. Die Benutzungsoberfläche stellt die Schnittstelle zum Benutzer dar. Sämtliche von ihm getätigten Eingaben werden von diesem Modul verarbeitet. Durch Funktionsaufrufe in anderen Modulen wird die vom Bediener gewünschte Aktion durchgeführt.

Jedes der Module verwaltet einen Teil der Datenstruktur. Die Modellierung der Datenstruktur, die zur Lösung der gestellten Aufgabe am besten geeignet erscheint, wird von jeder Arbeitsgruppe eigenständig vorgenommen. Die Zugriffe auf die Datenstruktur werden dabei nur innerhalb des Modules vorgenommen. Jedes Modul bietet lediglich eine Reihe von Funktionen als Schnittstelle zur Außenwelt an. Diese Funktionen werden explizit definiert und können von anderen Modulen zur Realisierung der gestellten Aufgaben aufgerufen werden. Dabei sollen möglichst wenig Informationen ausgetauscht werden. Die Arbeitsweise der Schnittstellenfunktionen soll für das aufrufenden Modul transparent sein.

Eine nachträgliche Überprüfung unseres Softwareentwurfs auf Modularität ergab, daß Kriterien wie modulare Zerlegbarkeit, Kombinierbarkeit, Verständlichkeit und Geschütztheit erfüllt wurden. Da spätere Änderungen in der Spezifikation, die häufig nach Diskussionen im Plenum notwendig wurden, lediglich Änderungen in wenigen Modulen hervorriefen, zeigte sich, daß auch das Kriterium der modularen Stetigkeit erfüllt wurde. Eine ausführliche Erläuterung dieser Kriterien als auch des modularen Softwareentwurfs findet sich in [Mey90].

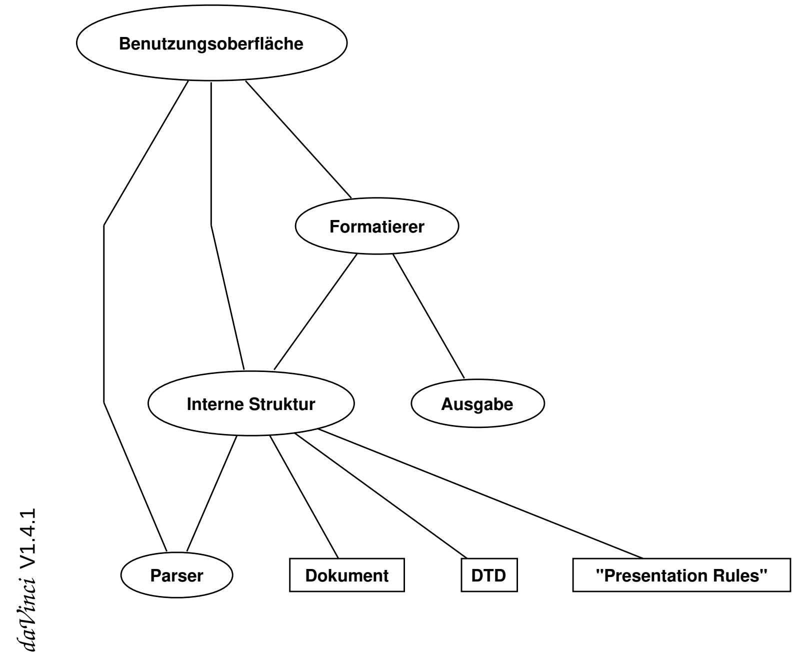

Abbildung 4.1: Aufbau des Programms

4.2 Interne Struktur

Achim Mahnke

Der Systemteil „Interne Struktur" (kurz *IS*) findet seine Keimzelle in unserem Dokument–Modell, das eine klare Trennung zwischen der logischen Struktur und der physikalischen Repräsentation eines Schriftstückes vornimmt. Die IS beinhaltet insbesondere den logischen Aufbau des Dokumentes und stellt darauf notwendige Operationen zur Verfügung.

Prinzipiell liegt die Interne Struktur zwischen dem Parser und dem Formatierer. Der Parser hilft bei der Übersetzung der externen Form des nach SGML–Standard ausgezeichneten Dokumentes in die interne Darstellung. Der Formatierer erhält Informationen über das Dokument und generiert ein gesetztes Abbild des Textes. Die Interne Struktur bietet für diesen Zweck Funktionen und Datenstrukturen an, um auf das logische Dokument bezug zu nehmen. Auch die Benutzungsoberfläche muß auf der logischen Struktur arbeiten; speziell Einfüge- und Lösch–Operationen werden von ihr benötigt und von der IS implementiert.

Neben dem Dokument selbst verwaltet dieser Systemteil auch die zugehörige Dokument–Typ–Definition (DTD) und die daran orientierten Präsentationsregeln. Der Aufbau und Zugriff auf diese Teile ist — streng nach dem Prinzip der abstrakten Datentypen — vollständig in der IS gekapselt.

Durch die Gegebenheiten der funktionalen Programmiersprache ASpecT (es gibt keine globalen Variablen, wie z.B. in LISP) ist die Interne Struktur auch Heimstatt aller Datenstrukturen, die von anderen Systemteilen, wie dem Formatierer oder der Benutzungsoberfläche für deren eigene Zwecke verwendet werden. Ihr jeweiliger Aufbau ist der IS vollständig unbekannt; sie definiert nur einen Datentypen — das Environment (`env`) — in dem alle Strukturen „gebeutelt" sind. Man kann also mit Fug und Recht vom „ForauS–Känguruh" sprechen...

Als weitere Komponente fand ein Modul mit ASpecT–Datentypen in die IS Eingang, die von mehreren Systemteilen benötigt werden, und aufgrund von programmiertechnischen Schwierigkeiten, die sich sonst ergeben würden (zyklische Importe u.ä.) hier am besten untergebracht sind. In diesem Modul ist außerdem die Funktionalität für die Fehlerbehandlung enthalten.
Der Zugriff auf die Interne Struktur ist für die anderen Systemteile nur über spezielle Schnittstellen–Module möglich. In ihnen sind jeweils die für einen Systemteil notwendigen Typen und Funktionen zusammengefaßt.

4.3 Dokument

Guido Frick

Die Gruppe Dokument als Untergruppe der Internen Struktur bestand ursprünglich aus Xiaoxia Lu und Guido Frick. Nach kurzer Zeit schied Xiaoxia jedoch aus, so daß nur noch Guido diese „Gruppe“ darstellte.
Aufgabe war es, einen in SGML ausgezeichneten Text in einer Datenstruktur abzubilden. Dabei ermöglicht SGML die Formulierung struktureller Beziehungen von Elementen/Objekten zueinander, Referenzierungen auf andere Objekte sowie Fließtext als Inhalt von Objekten. Hierzu wurden Funktionen benötigt, die eine solche Datenstruktur

- aufbauen
- manipulieren
- auslesen
- speichern.

Anfangs wurde überlegt, wie eine derartige Datenstruktur aussehen kann. Sie mußte eine baumartige Hierarchie von Objekten abbilden und Inhalte solcher Objekte speichern. Zusammen mit der Gruppe Formatierer, die ähnliche Datenstrukturen benötigte, wurde ein Datentyp gefunden, mit dem man schnell und effektiv auf einzelne Elemente zugreifen konnte. Nachdem nun die Datentypen gefunden waren, wurden elementare Zugriffsfunktionen entsprechend eines ADT implementiert.
Hierbei war es nötig, sich intensiv in die funktionale Programmiersprache ASpecT einzuarbeiten, was sich entgegen der Erwartung eines Gruppenmitgliedes als weniger problematisch erwies. Nach kurzer Zeit konnten bereits kompliziertesten Fehlermeldungskonstrukte mit Fassung ertragen werden.
Ein weiterer Punkt, der das Arbeiten der Gruppe erleichterte, war die Tatsache, daß das zu entwickelnde Modul ganz unten in der Aufrufhierarchie des Systems anzusiedeln war. So waren keine Funktionen anderer Module während der Entwicklung nötig, auf deren Fertigstellung man hätte warten müssen oder für die man *Dummy-Funktionen* erstellen mußte. Diese fehlenden Abhängigkeiten erleicherten ebenfalls das ausführliche Testen der Modul-Funktionen.
Dem Modul vorgeschaltet waren die Schnittstellenmodule der Internen Struktur, die Achim implementierte. Diese bildeten eine strenge Abgrenzung der IS-Module (Dokument, DTD, PR) zu denen des übrigen Systems. Für das Dokument ergaben sich Schnittstellen zum Parser (Einlesen, Speichern), zur Benutzungsoberfläche (u.a. Einfügen, Löschen eines Zeichens/Objektes), zur DTD (Kontextermittlung) und zum Formatierer (Objektinformationen). Nacheinander lieferten die einzelnen Gruppen ihre benötigten Schnittstellen. Anfangs wurde das Einlesen des Quelldokumentes durch den Parser ausformuliert, implementiert und getestet.

Während des gesamten Entwicklungsprozesses traten immer wieder kleinere Änderungen an den verschiedenen Schnittstellen auf, deren Durchführung jedoch in keinem Fall übermäßigen Aufwand zur Folge hatte.
Am Ende ergab sich eine Erweiterung der Schnittstelle zur Benutzungsoberfläche, da die logische Struktur des Dokumentes als Graph in dem Programm daVinci visualisiert werden sollte. Ergaben die Mehrzahl der implementierten Funktionen kaum sichtbare Ergebnisse, so lieferten diese Funktionen das, was ein Informatikerherz höher schlagen läßt: „verhältnismäßig" geringer Aufwand - großer Effekt.
Es muß klargestellt werden, daß es sich hierbei aber keinesfalls um triviale Funktionen gehandelt hat.
Da die Gruppe nur aus einer Person bestand, wurde annähernd 70% der Implementierung zu Hause vorgenommen, meistens bis spät in den Morgen. Die fehlerfreie Funktionsweise des Moduls wurde ausführlich mithilfe eines eigenen Testmoduls sichergestellt.
Durch die Arbeit der „Gruppe" steht nun ein Modul zur Verfügung, das stabil die erforderlichen Daten entsprechend der spezifizierten Schnittstellen verwaltet und ebenfalls die Möglichkeit der Erweiterung offen hält.

4.4 Formatierer

4.4.1 Einleitung

Tarik Ali, Andree Hähnel, Jan Hiller, Andreas Hinken, Bernd Rattey

Die Gruppe der Formatierer umfaßte einen großen Aufgabenbereich bei der Implementierung des ForauS-Systems. Vorrangiges Ziel war es, einen Formatierer zu schaffen, der hohen typographischen Anforderungen genügt und respektable Ergebnisse liefert.
Die Wurzel der Formatierer lag in der Gruppe Typographie (siehe Kap. 3.3), die bei der Feststellung der Anforderungen zum System viele wichtige Punkte gesammelt hatte. Aus der Untersuchung bereits bestehender Textverarbeitungssysteme ergab sich, daß sich ein Boxenkonzept für die Repräsentation eines Buchstabens anbietet. Eine Buchstabenbox umspannt ein einzelnes Zeichen und berücksichtigt Faktoren wie Ober- und Unterlängen einzelner Buchstaben. Geplant war, einzelne Boxen ähnlich einer Perlenkette an der Grundlinie aufzureihen. Diese Boxen sollten ihrerseits wieder in größeren Boxen zusammengefaßt sein, bis die Größe einer Seite erreicht ist (abgesehen vom Seitenkopf und -fuß). Schnell zeigte sich jedoch, daß ein solches Boxenkonzept, wie es auch vom Satzsystem TeX verwendet wird, für unsere Anwendung zu aufwendig sein würde. Wir haben folglich aus Effizienzgründen statt einzelner Buchstabenboxen, Wort- und teilweise sogar Zeichenstromboxen gewählt. Ein Vorteil dieser Vorgehensweise ist, daß ein großer Textblock mit gleichen Attributen mittels einer Funktion schnell dargestellt werden kann; die buchstabenweise Variante würde sicherlich ein Vielfaches an Zeit in Anspruch nehmen. Auch die Menge der zu speichernden Daten konnte durch diesen Ansatz gering gehalten werden, da Positionsinformationen nur zeilenweise gespeichert werden.
In der Fortsetzung der Arbeit aus der Typographie-Gruppe wurden die nötigen Parameter für die gegebenen Anforderungen festgestellt. Wir erkannten, daß in der Kürze der jetzt noch verbliebenen Zeit Anforderungen wie Fußnoten, laufender Titel und Referenzen nicht implementiert werden können. Die Anforderungen wurden daraufhin stark zusammengekürzt. Als Ergebnis steht jetzt ein Formatierer zur Verfügung, der die rudimentären Funktionen beherrscht und in der Lage ist, normalen Text zufriedenstellend links-, rechtsbündig und zentriert, sowie in Blocksatz darzustellen.

Um eine modulgerechte Implementierung zu erreichen, haben wir den Bereich der Formatierer in die Bereiche des HLF (High-Level-Formatierer) und der EF (Einfache Formatierer, das sind der Seiten- und Objektformatierer) aufgeteilt.
Die Aufgabe des High-Level-Formatierers ist es, eine Datenstruktur speziell für die Formatierer aufzubauen und zu verwalten. Weiterhin führt er einen ersten Lauf der Formatierung durch. Er beinhaltet das Ermitteln der Reihenfolge der Objekte und die Kapitelnumerierung. Ferner liefert er die Präsentationsregeln der internen Struktur; er stellt die zentrale Schnittstelle hierfür dar. In der Datenstruktur des HLF werden u.a. Informationen über die Formatierung und den Formatierungsstatus der Objekte gespeichert.
Die eigentliche Arbeit des Setzens wird von den Einfachen Formatierern geleistet. Sie setzen ausgehend von einem Objekt alle Zeilen eines Objekts und fügen diese zu einer Einheit „Objekt“ zusammen. Diese Objekte werden dann mithilfe des Seitenformatierers auf die Seiten des Dokuments aufgeteilt. Auch die inkrementelle Formatierung des Textes (es werden nur notwendige Formatierungen nach Veränderungen durchgeführt), wird von den Einfachen Formatierern erledigt, denn in dieser Ebene ist bekannt, wie groß ein Zeichen ist, und wieviel Platz in einer Zeile noch zur Verfügung steht.
Es existieren weitere Module des Formatierers. Darin sind z.B. Funktionen zur Ansteuerung der Ausgabe und zum Formatieren der Ausrichtung, welche erst in einem zweiten Formatierlauf durchgeführt wird, enthalten.
Die oben gemachten Ausführungen beschreiben die Konzepte des Formatierers auf sehr grobe Weise. Es wird dabei deutlich, daß es sich um einen sehr komplexen Vorgang handelt. Auf dem Weg zu unserem Ziel hatten wir eine Vielzahl von Ideen, deren Ausführungen den Rahmen dieses Beitrages sprengen würden.
Wir beschränken uns deshalb in den folgenden Seiten darauf, den Entwurf vorzustellen, der die Grundlage unserer Implementierung bildet.

4.4.2 Begriffe

Zunächst wollen wir einige grundlegende Begriffe definieren:

- **Logisches Element**
  Ein logisches Element ist ein Bestandteil der SGML-Struktur (z.B. „KAPITEL“). Jedem logischen Element ist eine eindeutige Typ-Id zugewiesen.

- **Textobjekt**
  Ein Textobjekt ist das Vorkommen eines logischen Elementes. Jedem Textobjekt ist eine eindeutige Objekt-Id zugewiesen.

- **Finales Element**
  Ein finales Element ist ein Textobjekt, das PCDATA (Buchstabenfolge in SGML, s.u.) enthält und explizit gesetzt werden muß. In unserer Datenstruktur sind die finalen Elemente die einzigen Elemente, denen intern Formatierungsattribute zugeordnet werden.

- **Nichtfinales Element**
  Ein nichtfinales Element ist ein Textobjekt, an dem in der Formatiererstruktur weitere Elemente angefügt sind. In einem Baum sind dies die Knoten, während die finalen Elemente die Blätter darstellen.

- **Physikalisches Element**
  Ein physikalisches Objekt ist ein durch den HLF erzeugtes Element. Es ist für die Ein-

fachen Formatierer transparent. Physikalische Objekte enthalten Informationen wie z.B. die Kapitelnumerierung.

- **Seitenlayoutparameter**
  Dies sind Angaben zum Aussehen einer Seite, wie z.B. die Seitenränder, Textbreite, Abstand zum oberen und unteren Rand usw.

- **Textlayoutparameter**
  Dies sind Angaben, die das Aussehen eines Textelementes (z.B. ein Fließtextabsatz) bestimmen (z.B. Schriftart, -größe, -schnitt usw.).

- **PCDATA**
  Hiermit werden die Zeichen bezeichnet, die gesetzt werden sollen - also der Text an sich.

4.4.3 Beschreibung der Formatierer

Wir unterscheiden beim Vorgang des Formatierens drei verschiedene Formatierer:

1. Den High-Level-Formatierer innerhalb des Formatierers (HLF intern)

2. Die Einfachen Formatierer (EF – Das sind der Seitenformatierer und der Absatzformatierer)

3. Den High-Level-Formatierer als Schnittstelle zu den anderen Moduln. Interaktion mit Moduln außerhalb des Formatierers (HLF extern).

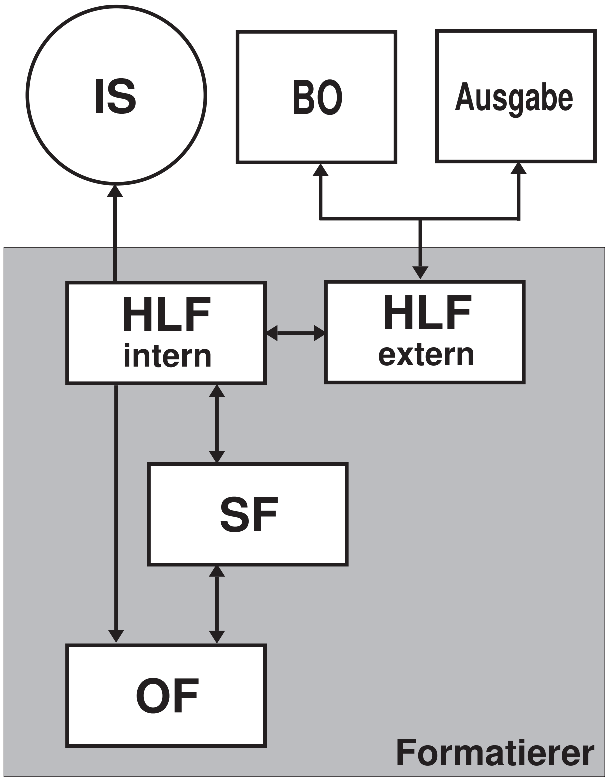

Abbildung 4.2: Gesamtübersicht: Die Interaktion des Formatierers mit anderen Moduln.

4.4.3.1 Der High-Level-Formatierer (HLF)

Da bei der inkrementellen Formatierung ein völlig anderer Modulaufruf als bei der ersten (initialen) Formatierung[1] abläuft, unterteilen wir die Beschreibung des HLFs in die Bereiche Intern und Extern.
Der HLF hat folgende Aufgaben zu erfüllen:

HLF intern

- Er erzeugt und verwaltet die Formatiererstruktur.
- Er ermöglicht den anderen Formatierern den Zugriff auf diese Struktur durch geeignete Funktionen.
- Er ist der Organisator innerhalb des Formatierers.
- Er führt den ersten Lauf der Formatierung durch.

HLF extern

- Er übernimmt die Kommunikation mit den Moduln außerhalb des Formatierers. Dies betrifft insbesondere das Ausgabemodul.

[1]siehe Abschnitt „Neuformatierung“

4.4.3.1.1 Der High-Level-Formatierer, intern Die Aufgabe des HLF ist es u.a. auf die Textobjekte der Internen Struktur zuzugreifen. Ebenso werden die Präsentationsregeln (PR) ausgelesen. Der HLF bestimmt die Reihenfolge der Textobjekte mit Hilfe der PR. Er liefert die finalen Elemente der so erzeugten Datenstruktur dann an die Einfachen Formatierer (EF). Eine solche Vorgehensweise impliziert, daß der HLF zunächst einmal das gesamte Dokument in die Formatierer-Struktur überträgt. Er führt einen ersten Lauf der Formatierung durch, bei dem die Kapitelnumerierungen u.ä. berechnet werden. Bei dieser initialen Formatierung erzeugt der HLF aus den Abhängigkeiten der Textobjekte aus der Internen Struktur eine eigene Struktur – die Formatiererstruktur. Hierbei handelt es sich um eine Baumstruktur. Auf diese Baumstruktur greifen die EF mittels geeigneter Funktionen zu.[2] Im Anschluß daran können die EF aufgerufen werden, um den Text zu setzen. In späteren Versionen soll der HLF zur Erzeugung von Verzeichnissen o.ä. aufgerufen werden können. Innerhalb des Formatierers sind nichtfinale Elemente nur dem HLF bekannt. Die Einfachen Formatierer bekommen vom HLF nur die finalen Elemente weitergereicht. Bis zu dieser Version werden Kapitelnumerierungen nicht von den EF sondern vom HLF selbst formatiert und gesetzt! Dies ist eine Ausnahme vom normalen Formatiervorgang.

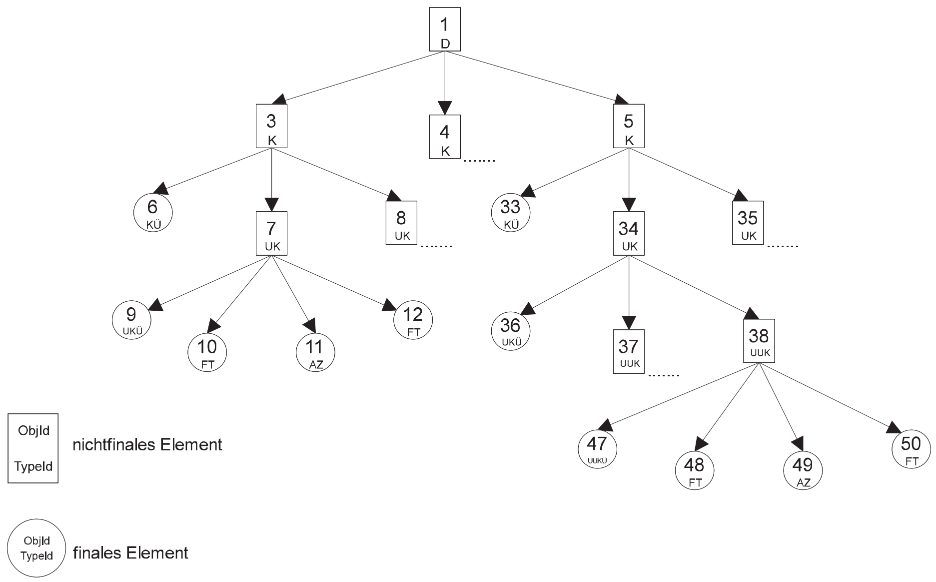

Abbildung 4.3: Ein Baum in der Formatiererstruktur.

Anhand dieses Beispiels wird die genaue Funktionsweise des HLFs erläutert. Zu Beginn teilt der HLF der Internen Struktur mit, daß er einen neuen Baum aufbauen möchte. Dann werden alle Textobjekte der ersten Ebene geliefert. In Abb. 4.3 entspricht dies dem „D“, was für Dokument steht. Falls mehrere Textobjekte vorliegen, muß mit Hilfe der Präsentationsregeln (PR) die Reihenfolge der Textobjekte festgelegt werden. Die Textobjekte werden nun in dieser Reihenfolge in den Baum eingefügt. Jedes neu eingefügte Textobjekt muß auf jeden Fall bei der späteren Formatierung direkt (finales Element) oder indirekt (nichtfinales Element) berücksichtigt werden. Daher enthält jeder Knoten und jedes Blatt ein Feld, das den Formatierstatus angibt. Für jedes gelieferte Textobjekt, das kein finales Element ist, werden wiederum rekursiv alle Textobjekte der Ebene (n+1) geliefert, was im Beispiel „K, K, K“ (was für Kapitel steht) ist.

[2] Die einzelnen Funktionen werden im Abschnitt „Modulbeschreibung Einfache Formatierer“ in der Dokumentation erläutert.

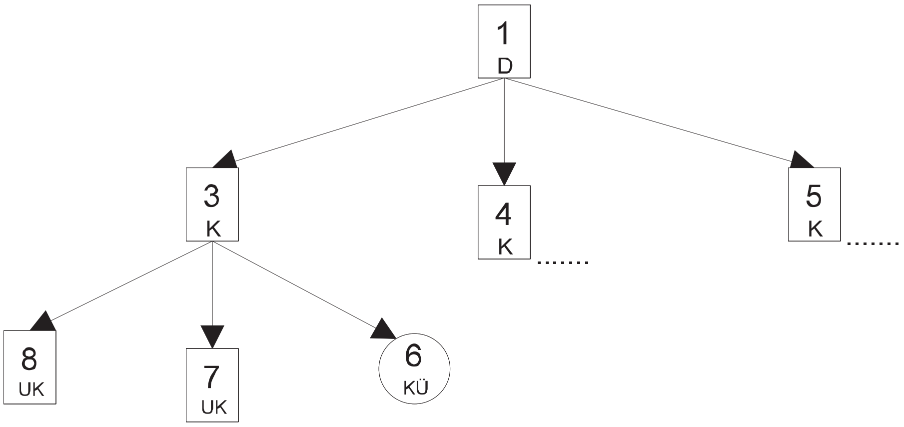

Abbildung 4.4: Eine Ebene des Baumes.

Im Beispiel wird für das Textobjekt „K" mit der Objekt-Id 3 als Ergebnis aus der internen Struktur „UK, UK, KÜ" geliefert (siehe Abb. 4.4). In den PR ist nun definiert, daß die Kapitelüberschrift (KÜ) vor dem ersten Unterkapitel (UK) erscheinen soll. Daher sind in der Formatiererstruktur diese Elemente anders sortiert als in der Internen Struktur. Das gilt für alle Textobjekte K dieses Beispiels.
Stößt der HLF beim Aufbau der Formatiererstruktur auf ein finales Element (im Beispiel auf das Textobjekt „KÜ" mit der Objekt-Id 6), so ist der entsprechende Ast komplett aufgebaut und das Rekursionsende erreicht.
Die zu den finalen Elementen gehörigen Attribute, die das Aussehen des Textes bestimmen, werden nicht direkt in der Formatiererstruktur gespeichert. Sie lassen sich über die Typ-Id aber jederzeit aus den in der internen Struktur gespeicherten Präsentationsregeln (PR) auslesen. Für den HLF spielt das Layout (also das Aussehen) des Textes zu diesem Zeitpunkt noch keine Rolle. Erst die Einfachen Formatierer fordern sich die Layoutparameter zu den finalen Elementen bei der eigentlichen Wort- und Absatzformatierung an.
Der HLF erzeugt auch sogenannte „physikalische Objekte". Dies sind z.B. die schon erwähnten Kapitelnumerierungen. Ein physikalisches Objekt kann vom Systembediener nicht direkt editiert werden. Es wird automatisch vom System durch den HLF erzeugt. Er reiht physikalische Objekte zwischen die anderen Textelemente ein.

4.4.3.1.2 Der High-Level-Formatierer, extern Während der interne HLF für den Aufbau und die Verwaltung der Formatiererdatenstruktur zuständig ist, organisiert der externe Teil das Zusammenspiel zwischen Moduln der Ausgabe und den Einfachen Formatierern.
Die Kernfunktionalität des externen HLF liegt in der Analyse der Dokumentveränderung durch den Benutzer und der Initierung entsprechender Formatierungsvorgänge durch die Einfachen Formatierer. Exemplarisch zeigen wir den Ablauf beim Einfügen eines Zeichens in den bestehenden, formatierten Text. Nachdem das Zeichen vom Benutzer entgegengenommen wurde, ruft die Benutzungsoberfläche den internen HLF auf. Dieser aktualisiert das sog. `changes`-Konstrukt, indem das Zeichen selbst und die aktuelle Cursorposition festgehalten wird. Anschließend fordert die Ausgabe vom externen HLF die Bildschirmveränderungen an. Hierzu analysiert der externe HLF durch das bereits gespeicherte `changes`-Konstrukt die angefallenen Änderungen: Ist ein Buchstabe oder Objekt eingefügt oder gelöscht worden? In diesem Fall kann die Position der Einfügemarke mit-

samt des Buchstabens ausgelesen werden. Jetzt wird die Formatierung der aktuellen Zeile durch die Einfachen Formatierer durchgeführt. Die als Ergebnis erhaltenen Parameter werden an die Ausgabe weitergeleitet und von ihr auf dem Bildschirm ausgegeben. Da die Einfachen Formatierer nur die Befugnis zur Veränderung der finalen Elemente haben, muß der externe HLF dafür sorgen, daß die nichtfinalen aktualisiert werden. Er aktualisiert in der Baumstruktur der Formatiererdaten evtl. geänderte Boxausmaße und Formatierungsstati.
Eine weitere häufig genutzte Funktionalität des externen HLF liegt in der Darstellung einer ganzen Seite. Da sich die Vorgänge vom oben beschriebenen unterscheiden, seien sie hier kurz umrissen: Zunächst wird überprüft, ob alle Objekte bis zum Seitenende formatiert sind. Ist dies nicht der Fall, wird eine entsprechende Formatierung veranlaßt. Wurde der Text bereits formatiert, wird eine Liste, die Wörter, deren Positionen und Auszeichnung enthält, aufgebaut und an die Ausgabe weitergereicht.
Schließlich sind im Modul des externen Formatierers noch einige Hilfsfunktionen für z.B. Postscript-Ausgabe und die Benutzungsoberfläche enthalten.

4.4.3.2 Die Einfachen Formatierer (EF)

Die Einfachen Formatierer (EF) unterteilen sich in drei Formatierer:

1. Objektformatierer (SimpleFormatter)
2. Seitenformatierer (PageFormatter)
3. Ausrichtungsformatierer (FormatAlignment)

4.4.3.2.1 Aufgabe der EF Der Objektformatierer nimmt sich ein finales Element und beginnt Wort für Wort einen Absatz zu formatieren[3]. Hierbei wird iterativ solange ein Wort hinter das andere „gesetzt“ (formatiert), bis das finale Element „leer“ ist. Die Layout-Parameter für ein zu setzendes (finales) Element bekommt der Objektformatierer über den HLF aus der Internen Struktur bzw. den Präsentationsregeln. Der Objektformatierer formatiert ohne Rücksicht auf Ausrichtung oder Seitenlängen. D.h. er formatiert ganze Boxen linksbündig, unabhängig davon, ob sie laut Präsentationsregeln in einer anderen Ausrichtung gesetzt werden sollen oder ob sie noch auf eine Seite passen.

Der Seitenformatierer hat nun die Aufgabe, die durch den Objektformatierer erzeugten Boxen auf einer Seite anzuordnen und im Bedarfsfall einen Seitenumbruch durchzuführen. Hierbei werden die Boxen ggf. in sog. Teil-/Subboxen aufgebrochen, wenn sie nicht komplett auf eine Seite passen oder die Seite schon vorher nahezu voll war. Die Teilboxen werden dann auf die nachfolgenden Seiten des Dokuments „verteilt“. Die beiden Einfachen Formatierer bekommen Textobjekte immer in der Reihenfolge, in der sie gesetzt werden. Im Speziellen sollte der Seitenformatierer die Kopf- und Fußzeilen sowie die Fußnoten setzen. Die hierfür notwendigen Informationen wären vom HLF ermittelt worden. Die Funktionalität ist allerdings noch nicht implementiert.

4.4.3.2.2 Ablauf der EF Die Einfachen Formatierer setzen alle finalen Elemente, die in der Formatiererstruktur des HLF gespeichert sind. Dazu muß der Seitenformatierer die Seitenlayoutparameter ermitteln. Der Objektformatierer erfragt dann für jedes finale Element über Funktionsaufrufe an den HLF die Objektparameter. Der Text des finalen Elements wird nun gemäß der

[3] Da unser System bis jetzt keine Silbentrennung beherrscht, ist das Wort die kleinste Einheit, die der Objektformatierer formatieren kann.

Text- und Seitenlayoutparameter auf mehrere Zeilen gebrochen und ausgerichtet. Die Umbruchstellen und die Größe und Lage der entstandenen Boxen werden dann gespeichert.
Ein Dokument besteht aus einer Liste von Textobjekten, wobei die Einfachen Formatierer nur die finalen Elemente setzen. Die nichtfinalen Elemente dienen lediglich der Repräsentation der Formatiererstruktur durch den HLF[4].

4.4.3.3 Ablauf des Formatierens

Wenn der Formatierer aufgerufen wird, gilt es zwei Zustände zu unterscheiden:

1. Neuformatierung
2. Inkrementelle Formatierung

Das Aufrufen des Formatierers geschieht jeweils durch das Ausgabemodul bzw. die Interne Struktur. Es/Sie stößt den Formatiervorgang durch den HLF (intern) an; dieser wiederum den Seitenformatierer. Letzterer ruft dann den Objektformatierer[5] und den Ausrichtungsformatierer auf.

4.4.3.3.1 Neuformatierung Bei der ersten Formatierung eines Dokuments ist die Datenstruktur des HLF leer. Daher wird zuerst der HLF aufgerufen, damit er ein Dokument initial formatiert und dabei die Formatiererstruktur aufbaut. Anschließend wird der Seitenformatierer aufgerufen, um die finalen Elemente vom Objektformatierer formatieren zu lassen. Die vom HLF erzeugte Datenstruktur wird dem Seitenformatierer dabei übergeben. Der Seitenformatierer richtet seine Anfragen nun an den HLF und übergibt jedesmal die komplette Datenstruktur an den HLF. Nach Beendigung der Formatierung durch den HLF und die Einfachen Formatierer wird die Struktur durch den HLF an die darüberliegende Funktion zurückgeliefert.

4.4.3.3.2 Inkrementelle Formatierung Die inkrementelle Formatierung soll das System richtig benutzbar machen. Selbstverständlich darf beim Einfügen eines Zeichens durch den Benutzer nicht jedesmal eine komplette Reformatierung des gesamten Dokuments durchgeführt werden. Das würde viel zu viel Zeit in Anspruch nehmen. Wir versuchen also nur die Teile neu zu formatieren, die von der Änderung (Einfügen eines einzelnen Zeichens) betroffen sind.

Die inkrementelle Formatierung geht wie folgt von statten:
Die Benutzungsoberfläche teilt dem internen HLF mit, was sich im Dokument in welchem Umfang wo geändert hat. Der interne HLF trägt diese Information in die Formatiererstruktur ein, reformatiert aber noch nichts. Im Anschluß ruft die Benutzungsoberfläche die Ausgabe zum Aufbau der veränderten Seite auf. Die Ausgabe leitet diese Anfrage sofort an ihre Schnittstelle zu den Formatierern (externer HLF) weiter. Hier wird überprüft, ob sich auf der entsprechenden Seite etwas verändert hat. Da dies der Fall ist, wird die Reformatierung des geänderten Bereichs (z.B. einer Zeile oder eines finalen Elementes) und der erneute Aufbau der Seite angestoßen.

Wir erhoffen uns dadurch, daß das System den Text während der Eingabe von Zeichen durch den Benutzer setzen und anzeigen kann, ohne daß unzumutbare Verzögerungen bei der Anzeige des Textes entstehen. Anderfalls wäre das System unbrauchbar.

[4] Nichtfinale Elemente werden im Abschnitt „HLF“ beschrieben.
[5] siehe auch „Die Einfachen Formatierer (EF)“

4.4.4 Schlußwort und Ausblick

Wir möchten die abschließende Betrachtung des Implementationsgrades der Formatierergruppe mit einem Rückblick beginnen: Erst im Herbst 1993 reifte nach den Vorbildern von FrameMaker und TEX die Idee einer boxorientierten Repräsentation des Textes. Wir hatten uns auf das abstrakte Modell der interagierenden Formatierer geeinigt, welches die strikte Aufgabentrennung eines Objekt-, Seiten- und übergeordneten Formatierers unter Berücksichtigung des Geheimnisprinzips vorsah. Wieder und wieder versuchten wir eine vollständige und schlüssige Schnittstellendefinition mitsamt der dazugehörigen Datentypen aufzustellen. Doch je mehr wir in die Materie eindrangen, desto mehr Fallstricke boten sich uns. In dieser Phase der Überspezifizierung kam uns der Zeitdruck des nahenden Projektendes zugute, der uns zur baldigen Implementierung zwang. Nachträglich bleibt anzumerken, daß dies in jener Phase die einzig richtige Entscheidung gewesen ist, da viele der erörterten Probleme durch Umstrukturierungen letztendlich nicht mehr zum Tragen gekommen sind.
Der Zeitraum von Januar bis Mai 1994 ist somit als Kernzeit der Implementierung anzusehen. In ihm reiften die Konzepte der Textrepräsentation nach dem Drei-Boxenkonzept, die Übertragung geänderter Bildschirmbereiche und die Berechnung relativer Bildschirmpositionen.
In der letztgenannten Funktion lag dann auch eine unserer konzeptionellen Schwächen. Da der Objektformatierer nur für die Zeilenumbrüche ausgelegt war und auch nur diese in der Formatiererdatenstruktur gespeichert werden, war es nicht ohne weiteres möglich, die Position eines *einzelnen* Wortes auf dem Ausgabemedium zu bestimmen. Doch genau diese Werte mußten wir für die Ausgabegruppe bereitstellen. Der Aufwand zum Ermitteln der Objektabhängigkeiten zwischen auf einer Zeile liegenden Objekten im Blocksatz war dann auch so komplex, daß die Funktionalitäten in ein eigens hierfür entwickeltes Modul `FormatAlignment` migriert wurden. Hier wäre ein Verbesserungvorschlag, die Verarbeitungsgranularität für den Objektformatierer auf Wortgröße herabzusetzen und zusätzlich noch die Position eines jeden Wortes in der Datenstruktur zu speichern. Dies würde gerade beim Bearbeiten eines Dokumentes eine enorme Geschwindigkeitssteigerung bewirken.
Eine weitere konzeptionelle Verbesserung wäre die Fokussierung der HLF-internen Zugriffsfunktionnen nicht auf den `final` bzw. `nonfinal` Status, sondern viel allgemeiner auf die Objekt-Identifikationsnummer. Gerade bei der Programmierung der Funktionalität des externen HLFs wurden allgemeine Funktionen zum Traversieren der Formatierbaumstruktur benötigt, die sich allein an der Objekt-ID orientieren. Als Notlösung für diesen Mißstand wurden Heerscharen von eigenen Funktionen geschrieben, die diesen Anforderungen genügen, in dem sie je nach Objekt-Status auf die finale oder nicht-finale Zugriffsfunktion verweisen.
Doch auch grundlegende Konzepte mußten überdacht werden: Jedem von uns war klar, daß der HLF derjenige sein würde, der die Verwaltung (sprich Aufbau und Aktualisierung) der selbst erzeugten Objekte, wie z.B. Kapitelnumerierungen, Aufzählungen, usw. übernehmen müßte. Bei der Implementierung fiel auf, daß es aber keine geeigneten Objekt-Arten für diese Zählobjekte geben würde. Dies war der Entstehungszeitpunkt der sog. physikalischen Objekte. Sie sind durch die Schnittstelle völlig transparent zur unteren Schicht und deswegen nicht für die Einfachen Formatierer als solche sichtbar. Trotzdem hat dieses Konzept Haken und Ösen, denn jetzt sind die durch den HLF erzeugten Objekte nicht mehr von den Einfachen Formatiern setzbar. Das Setzen der physikalischen Elemente durch den HLF ist aber ein glatter Bruch mit dem Konzept der Trennung der Zugehörigkeiten.
Diese Aufzählung erhebt nicht den Anspruch auf Vollständigkeit, sie soll vielmehr exemplarisch in unsere Implementierungsphase Einblick gewähren und Lösungsmöglichkeiten aufzeigen.
Da bei der Bewältigung der oben angerissenen Probleme und auch bei der komplexen Fehlersuche

mit den sehr rudimentären Mitteln des ASpecT-Debuggers viel Zeit benötigt worden ist, haben wir nicht den vollen Funktionsumfang der Formatierergruppe verwirklichen können. Funktionen, wie Seitennumerierungen, Fußnoten, Referenzen, Verzeichnisse und automatische Worttrennungen können erst in zukünftigen Versionen implementiert werden. Insgesamt sehen wir unsere Arbeit trotzdem als einen Erfolg an. Obwohl wir es nicht in allen Punkten geschafft haben, unsere Ziele zu erfüllen, konnten wir jedoch einen nach unseren Konzepten gestalteten Formatiererkomplex in verhältnismäßig kurzer Zeit implementieren. Die Module verhalten sich erwartungskonform und bedienen, die ihnen innewohnende Funktionalität, mit einer für dieses Modell optimierten Geschwindigkeit.
Es bleibt für uns abzuwarten, ob geeignete Diplomarbeitsthemen gefunden werden können, die u.a. die Weiterentwicklung nach o.g. Richtlinien forcieren. Das Interesse und die Spannung auf die so erzielbaren Ergebnisse wäre bei uns auf jeden Fall vorhanden.

4.5 Ausgabe

Bernd Rattey, Guido Frick

Die Ausgabe von ermittelten, errechneten, mit viel Mühe und Aufwand durch eine Vielzahl von Algorithmen geschleuste Daten auf dem Bildschirm, kann für einen Informatiker der Lohn für so manche Programmierstunde sein. Umso ärgerlicher war es, daß die ursprünglich hierfür eingeteilte Gruppe diesen Bereich unseres Systems aus an dieser Stelle nicht näher erläuterbaren Gründen nicht so recht bereitstellen konnte. Nach vielen vergangenen Plenumswochen wurde dann die Gruppe neu besetzt. Bernd und Guido bildeten nun die Ausgabe, wobei beide parallel noch in anderen Gruppen aktiv waren. So kam es, daß die Arbeit erst Ende Februar aufgenommen werden konnte.
Die Aufgabe lag darin, die vom Formatierer aufbereiteten Daten, von der Benutzungsoberfläche angeforderte Markierungen und Cursorpositionierungen auf dem Bildschirm als Teil der X-Applikation darzustellen. Die Ausgabe sollte während des Ablaufes der X-Applikation von der Gruppe Benutzungsoberfläche aufgerufen werden, die aktuellen Daten von dem Formatierer anfordern und das Dokument auf Grundlage der Daten darstellen. Die Ausgabe stellt somit die Schnittstelle für die Bildschirmausgabe dar und umfaßt im wesentlichen die folgenden Funktionsbereiche:

- Text in unterschiedlichen Schriftarten, -größen und -schnitten darstellen
- Bildschirmbereiche löschen, verschieben und invertieren
- Cursor darstellen und löschen

Aufgrund des späten Anfangs hatten wir die angenehme Situation, daß die Anforderungen an die Gruppe bereits klar umrissen waren. Da die Daten der Formatiererstruktur so gestaltet worden waren, daß sie auf die Bedürfnisse der Ausgabe zugeschnitten waren, konnte sehr schnell mit der Implementierung begonnen werden.
Zuerst wurden die verschiedenen Schnittstellenmodule zum Formatierer und der Benutzungsoberfläche ausformuliert und die aus den Aufgaben ergebenden Datenstrukturen der Ausgabe festgelegt. Hierbei stellte sich heraus, daß die systemnahen Funktionen, die vorwiegend mit dem X-System arbeiten mußten, vorzugsweise in C zu implementieren seien. Die höheren Funktionen sollten selbstverständlich in ASpecT programmiert werden, die dann die niederen Funktionen der C-Ebene verwenden würden. Dies führte dann zu der Aufteilung innerhalb der Gruppe, einer war für den C-, der andere für den ASpecT-Teil zuständig.

Das Hauptproblem lag nun darin, daß für die eigentlichen X-Funktionen hauptsächlich nur die Unix-Manuals als Anleitung zur Verfügung standen. Diese waren weder didaktisch gut aufgebaut, noch vermittelten sie die übergreifenden Dinge. Aus diesem Grund ist der C-Teil auch nicht unbedingt als „elegant" zu bezeichnen. Die von ihm geforderten Funktionalitäten liefert er jedoch.
Ein zweites Problem war anfänglich, daß wir nicht konsequent genug versucht hatten, so viel wie möglich in ASpecT zu programmieren. Die Umstellung kostete ein wenig Zeit und Nerven, stellte sich aber im nach hinein als lohnend heraus. Auch mußte die zuerst ausformulierte Cursorsteuerung grundlegend überarbeitet werden, da diese an verschiedenen Stellen nicht ausreichend flexibel war (Probleme bei Seitenwechsel und Neudarstellung einer Seite). Die neuen Funktionen stellten dann ebenfalls noch eine Vereinfachung der Schnittstelle dar.
Nach wenigen Wochen war die erste Phase der Implementierung abgeschlossen. Die Tests mit der Formatierergruppe verliefen dann sehr gut, für die Anbindung an die UI galt das nicht uneingeschränkt, aber auch hier konnten alle Probleme beseitigt werden.
Fazit: Die Gruppe konnte alle von den anderen Gruppen gestellte Anforderungen an sie erfüllen. Die Arbeit ging zügig voran, es gab wenig Komplikationen und wenige unschöne Momente.
Insgesamt kann man sagen, daß wir aus unserer Sicht den Ablauf in der Gruppe als sehr positiv bezeichnen. Wäre der Ablauf in anderen Gruppen ähnlich gewesen, so wäre das Projekt sicherlich ein wenig entspannter abgelaufen. Wir sind uns aber auch im Klaren darüber, daß die äußeren Umstände für die Gruppe sehr gut waren.

4.6 Parser

Christian Schaefer

Die Parsergruppe hatte die Aufgabe, Routinen für die Parser und den Generator zu entwickeln, mit denen es möglich sein sollte, Texte mit logischer Struktur und textuellen Attributen durch Präsentationsregeln (PR) in das $\mathrm{F}^{\mathrm{or}}\mathrm{a}^{\mathrm{u}}\mathrm{S}$-System einlesen und speichern zu können. Weiterhin sollte die Gruppe einen Weg aufzeigen, wie diese Werkzeuge in das Gesamtsystem zu integrieren seien.
Den Anfang der Entwicklungsarbeit bildete ein Arbeitsauftrag vom 21.10.93, in welchem die Gruppe untersuchen sollte, inwieweit SGML (Standard Generalized Markup Language) den Anforderungen, welche durch die Gruppe „logischen Struktur" definiert worden waren, genügt. Das Ergebnis dieser Untersuchung bildet die Aussage, daß SGML den vollen Anforderungskatalog der logischen Struktur erfüllt und darüber hinaus noch weitere Features bietet. Da laut Anforderungsdefinition der Gruppe „logische Struktur" nicht der volle Leistungsumfang von SGML benötigt wurde, bestand nun die Aufgabe der Gruppe darin, ein anforderungskonformes SGML-Subset zu erarbeiten. Diesen stellte die Parsergruppe dem Plenum am 28.10.93 vor. Gleichzeitig wurde die Möglichkeit des Einsatzes von generierenden Werkzeugen wie z. B. der Karlsruher-Toolbox [6] oder der UNIX-Werkzeuge Lex und Yacc erörtert. Die Gruppe und das Plenum sprachen sich für den Einsatz der Karlsruher-Toolbox als generierendes Werkzeug aus. Dieses Tool benötigt als Eingabe die Backus-Naur-Form (BNF) der zu erkennenden Zielsprache und generiert daraus einen C-Quelltext für den Parser.
In diesem Zusammenhang standen die nachfolgenden Bemühungen der Parsergruppe, auf bereits bestehende SGML-Sprachbeschreibungen in BNF-Form zurückzugreifen. Nachforschungen im Usenet innerhalb der SGML-Gruppe förderten eine brauchbare SGML-BNF aus den USA zu Tage. Bei dem Versuch, diese BNF Subset-konform zu gestalten, mußte die Gruppe feststellen,

[6] Projekt Compilerbau; Gesellschaft für Mathematik und Datenverarbeitung mbH, Karlsruhe 1992

daß der Umfang der vorzunehmenden Änderungen gleichbedeutend mit der Erstellung einer eigenen SGML-BNF käme. Große Probleme bei dem Versuch, einen Subset-konformen SGML-Parser zu generieren, führten auf Grund der hohen Komplexität der Sprache SGML, und der Eigenschaft, nicht LR (von-links-nach-rechts mit Rechtsableitung in umgekehrter Reihenfolge) parsbar zu sein, zu der Erkenntnis, daß ein Generieren des Parsers mit dem von uns gewählten Tool nicht möglich sei.
Deshalb führte die Parsergruppe am 18.11.1993 ein beratendes Gespräch mit Prof. Dr. Ute Bormann und Dr. Ing. Carsten Bormann aus unserem Fachbereich. Ergebnis dieses Gesprächs war der Hinweis auf entsprechende Spezialliteratur und den frei verfügbaren SGML-Parsers (SGMLS) von James Clark[7], der eine LR-parsbare Ausgabe liefert. Da nun ein Lösungsweg für das Hauptproblem, das Parsen von SGML-Texten, aufgezeigt war, konnte die Gruppe an die Aufgabe herangehen, benötigte Schnittstellen mit anderen Systemkomponenten, wie z. B. der Benutzungsoberfläche oder der Internen Struktur, abzusprechen.
Im Vordergrund der nun folgenden ersten Implementierungsphase stand die Fertigstellung des PR-Parsers, um Systemkomponenten, wie z. B. die der Internen Struktur, die in der Implementierung relativ weit fortgeschritten war, die Möglichkeit zu bieten, ihre Funktionalität mit real veränderbaren Werten testen zu können. Während dieser ersten Phase traten gehäuft Probleme im Zusammenhang mit der C-ASpecT-Schnittstelle auf. Diese beruhten zum größten Teil auf der unzulänglichen Dokumentierung der Schnittstelle, zum anderen auf kleinen ASpecT-Fehlern. Diese konnten Dank der bereitwilligen Hilfe unserer Projektbetreuer Michael Fröhlich und Mattias Werner schnell aus dem Weg geräumt werden. Da zu diesem Zeitpunkt die Parsergruppe als einzige Erfahrungen mit der C-ASpecT-Schnittstelle gesammelt hatte, übernahm sie die Aufgabe, Hilfsmodule zu entwickeln und den anderen Gruppen Hilfestellungen bei der C-ASpecT-Anbindung zu geben. In dieser Zeit entstanden auch die Hilfsmodule[8] `AviseC` und `TreasuresLib`, welche unter anderem auch von der Benutzungsoberfläche und der Ausgabe später benutzt wurden. Den Abschluß dieser ersten Phase bildeten die Semesterferien im Februar 94. Zu diesem Zeitpunkt waren alle Systemschnittstellen seitens der Parsergruppe definiert, und der PR-Parser war fertiggestellt.
In der nun folgenden zweiten Phase stand die Fertigstellung des SGML-Parsers im Vordergrund. Erste Tests mit dem frei verfügbaren SGML-Parser warfen neue Probleme auf. Auf Grund der hohen Komplexität dieses Parsers (mehrere tausend Zeilen Quellcode) wurde die anfänglich geplante Direktintegration zu Gunsten einer externen Integration, d. h. SGMLS läuft als eigenständiger Prozeß, fallengelassen. Hieraus ergaben sich nun Kommunikationsprobleme zwischen dem SGMLS und dessen Outputparser, welche mittels Pipes aber gelöst werden konnten.
Ein anderes Problem, das im direkten Zusammenhang mit der Integration von SGMLS in das ForauS-System steht, ist, daß SGMLS zwar nach dem Parsvorgang die ausgezeichneten SGML-Textelemente herausgibt, aber nicht die syntaktische Beschreibung des SGML-Dokuments (DTD = Document Type Definition), welche die Grundlage für eine korrekte Erweiterung, das heißt z. B. Einfügen, Ändern oder Löschen neuer Textelemente, des Dokuments bildet. Dieses Problem konnte in Zusammenarbeit mit dem Urheber von SGMLS per Email gelöst werden. James Clark überließ zu diesem Zweck der Parsergruppe das Tool DTDEL, welches zu 99% auf den Quelltexten von SGMLS basiert und die Aufgabe hat, die DTD eines SGML-Dokuments zu parsen und dessen syntaktische Beschreibung in geeigneter Form auszugeben. Auch dieses Tool wurde extern in das System integriert. Gegen Ende April war nun auch diese Phase mit Fertigstellung des SGML-Parsers beendet.
In der dritten und letzten Phase, die bis zum 13.6.94 dauerte, wurde von der Parsergruppe noch ein Generator zur Speicherung von SGML-Dokumenten inklusive der dazu gehörigen DTD aus dem

[7] Quelle: ftp.jclark.com pub/sgmls/sgmls-1.1.tar.Z
[8] vgl. ForauS-Programmdokumentation Kapitel 5 CTools.

ForauS-System heraus fertiggestellt. Des weiteren wurde noch eine Minimal-Postscriptausgabe implementiert, um so das System in seiner Funktionsweise abzurunden.

4.7 Benutzungsoberfläche

Rainer Schmidtke

4.7.1 Das dritte Projektsemester

Nachdem wir im vorangegangenen zweiten Semester in der „Anforderung an ein innovatives Dokumentenverarbeitungssystem“ ungefähr festgelegt hatten was unser Textsystem leisten sollte, folgten im dritten und vierten Projektabschnitt die Phasen der Entwicklung und Implementierung von ForauS. Dazu wurde zu Beginn des 3. Semesters das Semesterziel in Form von Meilensteinen konkretisiert. Sie sahen vor, eine erste lauffähige Programmversion mit minimalen Interaktionsmöglichkeiten bis Semesterende fertigzustellen.

4.7.1.1 Von der Anforderungsdefinition zum Systementwurf

Bevor die Implementierung beginnen konnte, mußte zuvor die Frage der Programmiersprache geklärt werden. Sie konnte in der vorlesungsfreien Zeit vor dem 3. Projektsemester zugunsten von ASpecT entschieden werden. Jedoch war C damit noch nicht verworfen. Systemteile wie der Parser, die Bildschirmausgabe und auch die Benutzungsoberfläche waren aufgrund von Werkzeugen bzw. Systemschnittstellen auf C angewiesen. Soweit wie möglich sollte die Programmkontrolle dennoch über ASpecT laufen.
Für die aus Michael, Son, Rainer und Volker gebildete Gruppe hieß es somit, daß Grundlagen bezüglich der Programmierung unter C, ASpecT (und deren C-Schnittstellen), als auch der Umgang mit dem X-System erlernt werden mußten. Die Anschaffung verschiedener Bücher und deren Durcharbeitung sollte uns also als erstes beschäftigen. Hierbei bemerkten wir schon recht früh, daß jedes Gruppenmitglied andere Voraussetzungen bezüglich Motivation und Vorwissen mitbrachte.
Das zunächst als Softwareergonomie-Richtlinien auf allgemeiner Ebene beschriebene mögliche Systemverhalten wurde zu einem Softwareergonomie - Funktionskatalog entwickelt. Themenschwerpunkte waren dabei u.a.

- die Informationsgestaltung am Bildschirm,
- die Selbstverwaltung des Systems,
- die Steuerungsmöglichkeiten des Benutzers,
- die Systemmeldungen und die Fehlerbehandlung.

Jede zunächst allgemein aufgestellte Funktion mußte auf unser Textsystem konkretisiert und dabei auch gleichzeitig die Realisierbarkeit abgeschätzt werden. In einer Abstimmung des Plenums gleich zu Semesterbeginn konnten Gewichtungen für die Implementierung gesetzt und anschließend mit den Ergebnissen der Gruppen „Standardfunktionalitäten“ und „Advanced Features“ in der „ForauS-Anforderungsdefinition“ festgehalten werden [For93].

4.7.1.2 Überschneidung von Aufgaben

Eher ungünstig in dieser Projektphase war die Involvierung von Gruppenmitglieder in zwei Arbeitsgruppen. Fast alle waren mit der Ausformulierung und der Adaption[9] der Präsentationsregeln und der Erstellung von sinnvoll einlesbaren Notationsbeispielen beschäftigt, so daß zunächst Ergebnisse der neu gebildeten Gruppe „Benutzungsoberfläche" ausblieben bzw. sporadisch Zwischenergebnisse aufgezeigt wurden. Auch die Absprache der Schnittstellen zwischen den einzelnen Moduln sollte darunter nur schleppend voran gehen.

4.7.1.3 Handlungsketten

Unabhängig von den anderen Gruppen beschäftigten wir uns parallel mit den möglichen und nötigen Handlungsfolgen bei der Erstellung von Texten mit einem Textsystem, um daraus Kenntnisse für den Entwurf zu gewinnen. Handlungsfolgen beschreiben die Reihenfolge der Eingaben, die der Benutzer für die Bearbeitung einer bestimmten (Routine-) Aufgabe bzw. Arbeitseinheit durch das System bedingt einhalten muß. Eine Arbeitseinheit ist zum Beispiel

- das Drucken des Textes mit Seitenangabe, eventuell Kopienanzahlangabe und Eingabe des Druckernamens,
- das Einstellen und Anpassen der Oberfläche (Cursor-Verhalten etc.),
- das Bearbeiten der Textelemente.

Diese Arbeitseinheiten sollten möglichst nicht durch verschachtelte Dialoge künstlich auseinandergerissen werden.
Die theoretischen Grundlagen dazu waren einem Teil der Gruppe schon aus Softwareergonomie-Vorlesungen bekannt:
Dem Benutzer sollte es ermöglicht werden, Handlungsziele zu bilden, einen Weg zum Ziel zu finden und die Handlung dann auszuführen. Dies setzt damit eine Überschaubarkeit der einzelnen Dialoge zwingend voraus. Das Ergebnis dieser Aktion mußte dann natürlich angezeigt werden, so daß diese überprüft werden können (Feedback; Weiteres siehe [Opp89]). Besonders das Arbeiten mit logischen Textstrukturen machte ein völlig neues Konzept nötig, welches erst noch zu konstruieren war.
Verschiedene Fragestellungen mußten Berücksichtigung finden:

- Es muß von einer Ausgangssituation ausgegangen werden, doch wie soll sie aussehen?
- Welche Handlungsfolgen sind von der Ausgangssituation vorstellbar oder bauen aufeinander auf?
- Was passiert mit den nicht von der Ausgangssituation ausgehenden Handlungen?
- Wie sind die Handlungsfolgen sinnvoll zu unterstützen?
- Welche Funktionalitäten sollen wie angeboten werden?
- Wie lassen sich diese Funktionen in - für den Anwender - logisch zusammenhängenden Gruppen fassen?

[9] Besonders die Parser- und Formatierergruppe kamen häufig mit neuen Änderungswünschen und Einschränkungen der Präsentationsregeln, die z.T. erst mit den fortschreitenden Entwurf von Algorithmen und Schnittstellen aufkamen.

Um diesen Fragestellungen nachzugehen, studierten wir typische X-Applikationen (Framemaker und einige X-Tools) und zogen [Sun90] hinzu.
Als Resultat dieser Überlegungen erhielten wir eine Auflistung von Benutzeraktionen, die dann in Form von festgelegten Dialogschritten in den Entwurf eingebracht wurden. Jedoch war uns zu diesem Zeitpunkt auch klar, daß dieser Entwicklungsstand bzw. die aufgestellten Handlungsmodelle nicht als endgültig definiert anzusehen war. Es kamen im Projektverlauf noch einige zusätzliche Anforderungen und Entwurfsänderungen bzw. -erweiterungen (wie z.B. die daVinci-Integration) auf uns zu, die in die bisherigen Ergebnissen eingefügt werden mußten. Das Grobkonzept mußte also einfach und flexibel ausbaubar sein.

4.7.1.4 Der Grobentwurf

Als nächstes konnten die bisherigen Erkenntnisse zu einen groben Entwuf von ForauS umgesetzt werden, der anschließend im Detail weiter auszubauen war.
Entsprechend dem „Open Look“ - Standard haben wir einen Informationsteil, einen Steuerungsteil (oder auch Kontrollbereich), einen Verarbeitungsteil und einen Meldungsteil untereinander vorgesehen. Die einzelnen Fensterabschnitte sind dabei automatisch optisch voneinander getrennt bzw. als zusammengehörend gekennzeichnet, um dem Betrachter dessen Erfassung zu erleichtern (Gesetze der Nähe und der Geschlossenheit aus der Softwareergonomie). Wir möchten deshalb im folgenden Text die Gestaltung dieser Masken- bzw. Fensterbereiche beschreiben.

4.7.1.5 Der Informationsbereich

Bei der parallelen Bearbeitung von mehreren Texten, als auch bei einem unaufgeräumten Desktop, mit einer Anzahl von Fenstern, kann der Benutzer schnell die Orientierung verlieren. Der Informationsbereich am oberen Fensterrand soll darüber Auskunft geben, welche Anwendung für dieses Fenster verantwortlich ist und welches Dokument gerade bearbeitet wird. Es enthält also neben dem Programmnamen (ForauS) auch den Dateinamen des gerade geladenen Dokumentes. Wurde noch kein Name für das Dokument vereinbart, so muß dieses ebenfalls kenntlich gemacht (zum Beispiel durch die Zeichenfolge „(NONE)“) werden[10].
Angedacht wurde zudem, den Namen der im Fenster angezeigten Sicht in dieser Zeile zu positionieren. Da jedoch bis zu diesen Zeitpunkt keine Sichten unterstützt werden konnten, wurde diese Idee nicht umgesetzt und später nicht mehr beachtet.

4.7.1.6 Gestaltung des Kontrollbereichs

Der Kontrollbereich oder auch Steuerungsbereich eines ForauS-Fensters ermöglicht dem Benutzer auf einfache Art und Weise, Kommandos auszuführen oder Programmeinstellungen zu erreichen. Dazu wurden verschiedene Menüobjekte im Haupt- oder Subfenster (Sichtfenster) vorgesehen, die zuvor reiflich überlegt sein wollten.
Es sind natürlich nur solche Funktionen anzubieten, die in der jeweiligen Arbeitssituation sinnvoll oder notwendig sind. Wir überlegten aufgrund der Handlungsanalyse, daß die Funktionen so in den Menüs zu verteilen sind, daß globale Funktionen (welche sich auf das gesamte System beziehen) im ersten Menü des Hauptfensters[11] liegen. Darauf folgend stehen Funktionen, die auf dem/die

[10] Diese Anzeige erregt aufgrund der Positionierung außerhalb des fovealen Wahrnehmungsbereichs (bei der Eingabe von Text ist der Blick des Betrachters wahrscheinlich auf den Arbeitsbereich oder auf die Tastatur gerichtet) nur wenig Aufmerksamkeit, so daß der Benutzer nur selten darauf achten wird.

[11] Zu diesem Zeitpunkt war uns die Notwendigkeit von Subfenstern bewußt, so daß auch diese berücksichtigt werden mußten.

Arbeitsbereich(-e) wirken. Erst danach planten wir in ihren Auswirkungen „kleinen“ Funktionen zum Bearbeiten des Textes in einem Menü einzubringen.
Vorgesehen wurden schließlich die vier Menütitel „Dokument“ (Document)[12], „Sicht“ (View), „Bearbeiten“ (Edit) und „Einstellungen“ (Properties), wie sie auch in den meisten anderen Anwendungen und X-Windows üblich waren.
Funktionen

- zur Ein- und Ausgabe,
- zur Steuerung des Anzeigebereichs,
- zur Manipulation von Textblöcken und Textelementen
- und zur Einstellung von anwendungsspezifischen Parametern

sollten durch die Menüleisten angeboten werden.
In einer konkreten Arbeitssituation (die Berücksichtigung eines aktuellen Kontext) nicht benötigten Funktionen können zu negativen Effekten (z.B. Irritationen des Benutzers) bis hin zur Ablehnung des Programms führen und sind deshalb unbedigt zu vermeiden gewesen. Dennoch konnten diese nicht einfach entfallen. Also planten wir solche Funktionen einheitlich („light“ hinterlegt) zu markieren, statt sie aus dem Menü zu entfernen.

4.7.1.7 Die Sichtenfenster

Ein weiteres Problem stellte sich uns mit der Frage der Funktionsaufteilung zwischen Haupt- und Sichtenfenster. Verschiedene Lösungskonzepte boten sich dazu an bzw. wurden von uns angedacht:

- Die Kontrolle könnte ausschließlich beim Hauptfenster liegen, d.h. die Subfenster enthalten keine Menüs oder Statusanzeigen.
- Ein extra vorzusehendes Kontrollpanel, wie es auch bei Framemaker zu finden ist, bietet alle notwendigen fensterübergreifenden Steuerfunktionen an.
- Alternativ bieten die Sichtfenster selbst adäquate Menüobjekte an, so daß zwar die Funktionen mehrfach angeboten werden, sie aber auch immer erreichbar wären.

Wir entschieden uns schließlich für ein Mischkonzept, wobei nur ein Fenster (das sogenannte Hauptfenster) Funktionen zur Ein/Ausgabe von Dokumenten und auch die Programmeinstellungen anbieten soll. Alle weiteren Fenster bieten nur die Möglichkeit, die eigene Arbeitsfläche lokal zu manipulieren.
Die Gründe dafür liegen auf der Hand: Ein direkter Bezug zwischen Funktionsaufruf und Programmreaktion sollte vorhanden sein und kann nur schwer hergestellt werden, wenn diese in (verschiedenen) Fenstern stattfinden.
Möglicherweise ist es notwendig, für unterschiedliche Sichten auch verschiedene Menüs bereitzustellen. Im Sinne größtmöglicher Flexibilität blieb uns also nur die gefundene Lösung.

[12] Die Bezeichnung „Dokument“ statt dem üblichen „Datei“ oder „File“ entspricht mehr der Intention und Wortschatz des Benutzers von Textsystemen und wurde dehalb vorgezogen.

4.7.1.8 Der Arbeitsbereich

Der Arbeitsbereich enthält den zu bearbeitenden Text[13] und ist für Haupt- und Sichtenfenster gleich ausgelegt. Er kann auf vielfältiger Weise beeinflußt werden:
Sicherlich mußte vorgesehen werden, den Text mit der Tastatur einzugeben und auch wieder zu löschen. Zudem war da ja noch der Text-Cursor. Er sollte die Textposition angeben, an der das nächste Zeichen positioniert wird und das Springen innerhalb des Textbereichs ermöglichen (textbezogene Cursorsteuerung). Blockmarkierungs- und Manipulierungsoperationen benutzen sowohl den Grafik- als auch den Text-Cursor und das auf sehr komplexe Art und Weise. Um dem Benutzer hierbei zu unterstützen, war sowohl die Tastatur, als auch der Grafikcursor für die Steuerung der Manipulation(-sart) vorgesehen. Hierbei waren allerdings die Schnittstellen zu den Formatierern, der Ausgabe und der internen Struktur stark gefragt, die bis dahin nur vage zu diesem Bereich formuliert waren.
Mit dem Schieber (Slider) neben dem Arbeitsbereich sollte der Darstellungsbereich verschoben werden. Zunächst wurde geplant, so im ganzen Dokument „blättern“ zu können (wie es zum Beispiel in Framemaker möglich ist), doch konnte der Formatierer nicht ständig alle Seiten formatiert vorrätig halten, so daß wir uns auf das Scrollen einer Seite begnügen mußten. In einer zukünftigen Version mag diese Idee jedoch noch aufgegriffen werden.

4.7.1.9 Der Feinentwurf

Nachdem das grobe Aussehen von ForauS feststand, war nun Detailarbeit zu leisten. Als erstes beschäftigten wir uns dazu mit Aktionen, die mit großer Sicherheit auch andere Gruppen tangierten, so daß diese recht frühzeitig mit der Umsetzung in Schnittstellen und Funktionen beginnen konnten bzw. Tests damit durchfürbar waren.
Das Markieren von Texten mußte aus den Anforderungen entwickelt werden, denn hieran gebunden sind Funktionen wie das Einfügen, das Löschen, das Ausschneiden und das Verschieben von Textteilen. Es war zu definieren, wie das Selektieren grundsätzlich durchgeführt werden sollte, aber auch ob und wie diese Operationen mittels Dialogfolgen unterstützt werden könnten. Ebenso war zu überlegen, ob neben der Maus auch die Tastatur dazu einzusetzen ist. Es stellte sich jedoch am Ende heraus, daß die komplette Unterstützung der Tastatur (mit allen Funktionstasten) doch zu viel Zeit in Anspruch neben würde. Sie sind deshalb nur teilsweise korrekt belegt.
Wir erkannten, daß das Arbeiten auf Zeichenketten und Elementen konzeptuell different durchzuführen ist. Fragen, wie

- Sollen Zeichenketten elementübergreifend markiert werden können?
- Wie kann dann die Textstruktur dabei erhalten bleiben bzw. kann dem Benutzer klar gemacht werden, warum einige Textpassagen nicht veränderlich (physikalische oder strukturgebundene Elemente) sind?
- Wie läßt sich ermitteln, wo welche Elemente erzeugbar oder löschbar sind und damit für die Blockmarkierung in Frage kommen?

wurden ausgiebig erörtert und die Realisierbarkeit der einen oder anderen Alternativen auch in gruppenübergreifenden Gesprächen diskutiert. Hieraus kam die verblüffende Erkenntnis, daß die zunächst einfach erscheinenden Blockoperationen auf Zeichenketten nur schwer für den Benutzer

[13] Alle Augenbewegungen sind während der Arbeit mit dem Programm auf dieses Feld konzentriert, so daß überflüssige bzw. vom Text abweichende Augenbewegungen minimiert werden.

transparent darstellbar waren und auch deren Implementierung einen hohen Aufwand erfordern würde.[14]
Der Entwurf zur Manipulation von logischen Elementen war dagegen wider Erwarten einfach. Steht der Cursor auf einem Element läßt sich aufgrund der vorgegebenen Dokumentenstruktur (DTD) entscheiden, ob und welches Element wo einfügbar ist[15]. Um dieses Element zu löschen boten sich mehrere Methoden an. Zum einen könnten - um die Dokumentstruktur konsistent zu halten - automatisch alle darunter oder darüber liegenden, abhängigen Elementen gelöscht werden. Dabei sind die löschbaren Elemente zu ermitteln und den Benutzer zur Bestätigung anzuzeigen. Zum anderen (und das erschien uns sinnvoller) könnte „manuell" vorgegangen werden. Alle abhängigen Elemente müssen zuerst vom Benutzer gelöscht worden sein. Ist dieses nicht der Fall, so kann das Element nicht selektiert bzw. nicht entfernt werden. Da das Kopieren und Verschieben von Elemente mit Löschen und Einfügen erreicht werden kann, war der Entwurf dazu abgeschlossen.
Eines mußten wir dennoch zugeben: Besonders benutzungsfreudlich waren die bisher geplanten Interaktionsmöglichkeiten nicht. Irgendwie sollte der Benutzer mitgeteilt bekommen, wo der Textcursor gerade steht und was er dort für Aktionen tätigen kann. Schließlich ist es nicht leicht, bei größeren Texten die Übersicht über alle Elemente zu behalten. Ein Meldungs- und Statusbereich mußte her!
Die erste Idee dazu traf nur wenig Zustimmung: Im Randbereich neben dem Text sollten die im Text auftretenden Elementnamen dargestellt werden. Da die Formatierer-Gruppe diese Aufgabe aufgrund des hohen Aufwandes nicht übernehmen wollte[16], konnte sich eine zweite Idee durchsetzen. Die für die Oberfläche Open Look üblichen Applikationen besitzen alle eine sogenannte Info-Zeile, in der wir den aktuellen Strukturpfad[17] und auch die Seitennummer anzeigen lassen können.
Bald wandten sich einige Gruppenmitglieder der Ausgestaltung von Menüs zu, die in Grundzügen ja schon vorhanden waren. Eine einheitliche und damit erwartungskonforme Benennung und Abfolge kann für den sicheren Umgang mit dem Textsystem nur dienlich sein. Wir orientierten uns also an den Vorgaben der Literatur.
Das Meldungsfenster war in erster Linie für die Ausgabe von Fehlertexten und Systemzustandsmeldungen vorgesehen. Dessen Potential ist damit lange noch nicht erschöpft. Schon sehr früh dachten wir darüber nach, wie Handlungen für den Benutzer zu planen und deren Durchführung zu unterstützen seien, als auch wie Verständnis für die Reaktion des Systems zu entwickeln waren (Transparenz). Hierzu bot sich ebenfalls das Meldungsfenster an, doch konnten diese Vorstellungen aus Zeitmangel nicht weiter verfolgt werden.
Schon am 5.10.93 konnten erste Vorschläge zum Aussehen von ForauS (Zunächst der Grobentwurf mit einzelnen Fragestellungen zum Feinentwurf) im Plenum präsentiert und diskutiert werden. Die konkrete Umsetzung unserer Ideen konnte beginnen, obgleich die Oberfläche natürlich noch in wenigen Details weiter ausgearbeitet werden mußte.

[14] Automatische Blockerweiterungen oder Blockreduzierungen bzw. die Prüfung des Selektionsbereichs bezüglich der Textstruktur ist, da dabei physikalische Zeichenreihenfolgen und logische Hierarchien berücksichtigt werden müssen, algorithmisch äußerst anspruchsvoll und im angemessenen Zeitrahmen (Programmier- und Programmablaufzeit) wohl nicht realisierbar.

[15] Zunächst hatten wir angenommen, daß der Cursor hinter oder vor einem Element stehen muß, um dort ein (neues) Element einzufügen. Später kam uns jedoch die Idee, dem Benutzer einen Dialog dazu anzubieten.

[16] Das Konzept des Formatierer-Kerns hätte für diese komplexe Formatierung komplett neu konzipiert werden müssen.

[17] Der ganze Pfad war zu lang, so daß wir uns auf das aktuelle Element beschränkten.

4.7.1.10 Erste Implementierungschritte

Auf Anraten der Projektleitung, die erste Version ausschließlich in C zu programmieren, begann die Implementierungsphase.[18]
Nur wenige von uns hatten zu diesem Zeitpunkt ausreichende C-Kenntnisse, so daß der Programmierstil als auch die Programmiergeschwindigkeit zu Anfang sehr differierten. Die Arbeit ging entsprechend schwerfällig voran. Auch die Programmierung des X-Systems war zunächst nur schwer zu beherrschen. Jedes kleine Objekt und jeder Zustand eines Objekts mußte beschrieben werden, d.h. es mußten die Objektnamen, deren hierarchische Abhängigkeit innerhalb von Objekten, als auch die Bedeutung aller relevanten Objektattribute modelliert werden. Ein Oberflächengenerator hätte uns hierbei viel Arbeit abnehmen können!
Nachdem die Frage der Modularisierung der Systemkomponenten ausgiebig im Plenum erörtert und auch entschieden wurde, konnten konkrete Schnittstellen zu den Modulen entworfen werden.[19]
Wir erkannten, daß es ungünstig ist, Ausgabetexte im Programmcode zu definieren. Auch die einzelnen Gruppen konnten nicht einfach beliebige Texte anzeigen. Die Konsistenz und Änderungsfähigkeit war so nicht zu gewährleisten. Zusätzlich war überlegt worden, doch noch englischsprachige[20] Dialoge vorzusehen. Ein neues - am besten mehrsprachiges - Konzept mußte also gefunden werden. Entschieden wurde, diese Texte in eine externe Datei zu definieren. Dazu boten sich grundsätzlich zwei Möglichkeiten an:

- Die Texte werden als „defines“ zum Quelltext hinzugefügt und über Bezeichner angesprochen, oder
- sie werden alternativ in Form einer Datenbankstruktur abgelegt, so daß der Text bei Programmstart zunächst mittels einer Importierfunktion gelesen werden muß.

Da die erste Alternative einfacher und schneller zu realisieren schien, machten wir uns sogleich an die komplette Überarbeitung des C-Quelltextes.
Wir stellten fest, daß das X-System von sich aus mehrere Sprachen unterstützt und die verschiedenen Texte dazu in Datenbanken mit eigenem Format erzeugt werden müssen. Untersuchungen ergaben allerdings, daß die aktuelle Installation des X-Systems die sprachliche Umschaltung nicht unterstützte, so daß diese Möglichkeit nicht weiter verfolgt wurde.
Parallel dazu wurden die Schnittstellen zu den anderen Moduln weiterentwickelt. Als erstes konnte unter Beteiligung aller Gruppen die Fehlerschnittstelle festgelegt werden. Sie war notwendig geworden, da die auftretenden Fehler hierarchisch tiefer liegender Funktionen irgendwie an das Benutzungsoberflächen-Modul übergeben und dann angezeigt werden sollten.
Darauf folgte die Definition der Schnittstellen zum Formatierer, dem Parser und der internen Struktur.
Ein Meilenstein der Entwicklung von ForauS konnte am 20.1.94 erreicht werden. Der erste Prototyp der Oberfläche (ausschließlich in C geschrieben) war erstellt.
Das nächste Ziel war die Anbindung von ASpecT und damit die weitere Nutzung der Schnittstellen der anderen Gruppen. Die bisherige Modulaufteilung der Benutzungsoberfläche war dabei aufgrund ASpecT-technischer Probleme nicht beizubehalten. Es entstand ein einziges Modul, welches ein paralleles Arbeiten sehr erschwerte.

[18] Diese Entscheidung stellte sich anschließend jedoch als nicht so günstig heraus, da die nachfolgende Anbindung an ASpecT immense Anpassungen erforderlich machten.
[19] Dieser Entwicklungsprozeß dauerte einige Zeit, so daß zunächst immer noch nur in C programmiert wurde.
[20] Ursprünglich war der Benutzerkreis auf den deutschsprachigen Raum begrenzt worden. Die Idee der internationalen Verbreitung unseres Systems wurde erst später geäußert.

Auch mußten die Funktionsköpfe und alle globalen Variablen auf ASpecT angepaßt werden. Nach den zum Teil recht aufwendigen Änderungen im C-Code war die ursprügliche Programmstruktur kaum noch zu erkennen. Nachteilig wirkte sich diese Umstellung auf die Übersetzungszeit von ForauS aus. Da sich unser Modul auf der höchsten Hierarchie-Ebene befand, waren lange Übersetzungszeiten zu erwarten. Sprunghaft stieg diese Zeit auf bis zu einer halbe Stunde an und erschwerte damit das Arbeiten mit ASpecT. Da insbesondere kleinere Änderungen an der Oberfläche zu testen war, machte das Arbeiten nur noch abends und am Wochenende Spaß!
In den nun folgenden Schritten mußten nach und nach alle ASpecT-Module eingebunden werden. Ein Anfang wurde mit dem Parser-Modul gemacht, da darauf aufbauend die Übernahme von Dokumentstrukturen in die interne Struktur angegangen werden konnte. Erst danach konnte das Formatierer- und Ausgabemodul eingebunden werden.
Am 3.3.94 war es schließlich soweit: Die Anbindung aller Module war geschafft. Jetzt fing jedoch die eigentliche Arbeit erst an. Einige Schnittstellen mußten verfeinert und prototypisch definierte Funktionen waren noch auszuprogrammieren und zu testen. Da das Oberflächen-Modul mit fast allen anderen Systemteilen zu tun hat, blieb bei uns auch die Aufgabe des Integrationstests. Gerade zum Projektende bedeutete dies einen besonderen Streß, da immer wieder Fehler in den Schnittstellen und in den einzelnen Modulen festgestellt werden konnten.
In der verbleibenden Zeit bis zu Ende der Semesters konnten weitere Funktionen implementiert werden: Fast alle Einstellungen der Benutzungsoberfläche waren nun ladbar. Sie umfaßten alle wesentlichen Fensterkoordinaten, Pfadnamen und besonders die Einstellungen unter dem Menütitel „Properties“.
Wir entwickelten die Funktion zur Erstellung neuer Dokumente. Diese bedingte die Auswahl einer DTD, so daß auch dafür ein Dialog anzubieten war. Auch konnte in Absprache mit der Internen-Struktur-Gruppe eine Liste der Sichten in Abhängigkeit des Dokumentes implementiert werden. Erstmalig war somit ein Sichtwechsel möglich, wurde allerdings noch von keinem Modul richtig unterstützt.
Damit war nun auch das Ende des 3. Projektsemesters erreicht. Eine Aufforderung zum pausieren sollte die folgende vorlesungsfreie Zeit jedoch nicht sein. Ganz im Gegenteil! Die Implementierung ging weiter voran. Dazu traf sich das ForauS-Plenum regelmäßig, so daß Probleme als auch Fragen in der Gruppe besprochen werden konnten.
Natürlich hatten nicht alle Gruppenmitglieder das gleiche Wissen über softwaretechnische oder software-ergonomische Vorgänge. Dies wurde bei der Implementierungsaufgabe sehr schnell deutlich. Schließlich war gerade das Arbeiten am Benutzungsoberflächenmodul aus unserer Sicht besonders anspruchsvoll. Die Programmiersprachen C und ASpecT, die zu berücksichtigenden Gestaltungsrichtlinien und Standards, die X-Windows-Strukturen und -Konzepte, Änderungswünsche anderer Teilgruppen, sowie das Einhalten der eigenen Planung mußten regelmäßig neu diskutiert werden. Die Defizite, die bei dem einen oder anderen vorhanden waren, konnten in der Gruppe relativ stark ausgeglichen werden. Was wir nicht selber wußten, schlugen wir in Büchern nach oder haben andere Projektteilnehmer befragt. Auch die Motivation innerhalb der Gruppe wechselte von Woche zu Woche, befand sich jedoch stets auf recht hohem Niveau. Trotzdem (oder vielleicht gerade aufgrund des hohen Niveaus) gelang es uns nicht, jeden in der Gruppe so zu fördern bzw. zu fordern, wie es eigentlich nötig gewesen wäre. Ein Gruppenmitglied wechselte deshalb zur Postscript-Ausgabe-Gruppe. Aber auch diese Erfahrung ist für uns sicherlich mehr als persönliche Bereicherung einzustufen — als das es eine Niederlage war.

4.7.2 Das vierte Projektsemester

Rainer Schmidtke

Das 4. und zugleich letzte Projektsemester stand weiterhin ganz im Zeichen der Implementierung. Schließlich war das geplante Ziel in den wesentlichen Punkten noch nicht erreicht. Besonders die Cursorsteuerung und die Interaktion auf den logischen Textelementen sollten bis Projektende noch ausgearbeitet werden.
Wöchentlich konnten über Programmerweiterungen im Plenum berichtet werden. Der PR-Parser war als erstes vollständig angebunden, so daß die zuvor schon ladbaren SGML-Texte theoretisch angezeigt werden konnten. Es folgten Funktionen zur Anzeige von Fehlermeldungen und zum Durchreichen von Tastatureingaben an die zuständigen Module.
Am Seminartag kamen neue Anforderungen auf uns zu. Zur Darstellung der Dokumentstruktur und zur Auswahl der logischen Elemente sollte das Graphsystem daVinci benutzt werden. Auch wurde die englische Sprache für alle Textausgaben festgelegt, so daß neue Aufgabe auf uns zu kamen.
Da wir eine Sprachumstellung im Entwurf schon eingeplant hatten, konnte in der darauf folgenden Woche die Umstellung auf englisch als abgeschlossen betrachtet werden. Doch auch andere Systemteile wurden eingefügt: Die Intervallspeicherung, die Textmodifikationserkennung, eine selektive Fehlerausgabe und sogar (von anderen Gruppen jedoch nie richtig unterstützt) verschiedene Sichtfenster.
Nach und nach konnte auch die Cursorsteuerung in den Griff bekommen werden. Nachdem die zuerst ausgearbeitete Schnittstelle sich als eher untauglich erwies, wurde im zweiten Anlauf diese zum Teil radikal geändert. Ebenfalls war das Konzept der inkremetellen Formatierung seitens der Formatierergruppe jetzt soweit implementiert, daß das Oberflächenmodul auch diese Schnittstelle unterstützen konnte.
Ein weiterer Meilenstein war mit dem korrekten Einfügen eines Zeichens an der Cursorposition erreicht. Das Löschen von Buchstaben folgte dann auch bald darauf. Doch je mehr Tasten nun benutzt wurden, je deutlicher wurde, daß dazu das letzte Wort nocht nicht gesprochen war. Die von uns an die ASpecT-Module weitergereichten Tastaturereignisse stimmten nicht mit den gedrückten Tasten überein. Hierfür schienen die benutzten C-Makros der X-Schnittstelle nicht die richtigen Werte zurückzugeben. Eine Umstellung der Betriebssystemparameter bzw. eigene Auswertung der Tastencodes war allerdings nicht sinnvoll, da entweder die anderen Applikationen (wie der Editor oder der Debugger) dann nicht mehr richtig bedient werden konnten, oder die Lauffähigkeit von ForauS auf anderen Rechnern (ohne geänderte Tastenzuordnung) anzuzweifeln war.
In den folgenden Wochen konnten nochmals einige Erfolge gefeiert werden. Das Einfügen und Löschen von Elemente, die (nicht leichte) Anbindung von daVinci und die Druckerausgabe über Postscript waren soweit fertig.
Fast zuletzt konnten schließlich auch die von den einzelnen Gruppen nur halbherzig erstellten Fehlermeldungen ausgewertet und dargestellt werden.
Zum Abschluß des Projektes folgte die Präsentation unserer Arbeit, als auch die Besprechung des Abschlußberichts.

4.7.3 Abschließende Betrachtung

Daß bei der Implementierung natürlich nicht alles reibungslos verlief und einige Fehlentwicklungen auftraten, hatte verschiedene Ursachen:
Besonders aus der Tatsache, daß die Entwurfsphase zeitlich stark reduziert[21] wurde, resultierte ein differierendes Modell vom Ablauf einzelner Vorgänge. Es war lange Zeit nicht deutlich ge-

[21] Wir hatten uns zu lange mit dem Sammeln von Anforderungen und der schriftlichen Ausarbeitung beschäftigt.

nung, welche Funktionalität wann und in welcher Koordinations-Reihenfolge bereit gestellt werden sollte.
Aber auch die terminliche Absprache von einzelnen Gruppen war ein schwieriger Prozeß. Als zuständige Arbeitsgruppe für das Hauptmodul fiel auf uns die Aufgabe, alle anderen Module zu koordinieren. Dieser Kommunikationsprozeß konnte nur dann in einem angemessenen Zeitrahmen erfüllt werden, wenn alle Beteiligten nahezu simultan ansprechbar sind. Leider war dieses oft nicht möglich.
Zuständigkeiten bei der Ausarbeitung von Teilproblemen waren oft nicht eindeutig verteilt, so daß daran beteiligte Gruppen entweder parallel Lösungen suchten, auf die man sich erst wieder einigen mußte (was viel Zeit in Anspruch nehmen konnte), oder Gruppen aufeinander warteten, mit der Vermutung, die andere Gruppe befasse sich schon mit einer Problemlösung.
Die gruppenindividuelle Festlegung von Prioritäten sei hierzu weiterhin aufgeführt. Alle Komilitonen hatten neben der Definition von Schnittstellen auch ihre eigenen, gruppenspezifischen Aufgaben, die hin und wieder vorgezogen wurden. So kam es dazu, daß wir auf andere Gruppen, als auch andere auf uns warten mußten.
Die bei der Programmierung auftretenden Umsetzungsschwierigkeiten der Schnittstellen hatten wir jedoch nicht vorausgesehen. So kam es dann bei der Implementierung zu Unstimmigkeiten über die Benennung von Funktionen oder den Parametern. Auch führten fehlerhafte, ungetestete Funktionen immer wieder zu zeitraubenden Suchaktionen[22]. Einzelne Funktionen wurden manchmal (so unser Eindruck) nicht ausführlich genug mit anderen Modulschnittstellen getestet.
Ein nicht zu unterschätzendes Problem ergab sich aus schon beschriebenen Schnittstellenumstellungen: Nicht jede Funktion ließ sich nach Absprache von Namen und Parametern unabhängig von anderen Gruppen entwickeln. Gleichzeitige Umstellungen von Funktionen wurden nötig, so daß alle beteiligten Personen anwesend sein mußten.
Zeitintensive Tests und die recht aufwendigen[23] Darstellungen von Zwischenergebnissen im Plenum reduzierten zusätzlich die Geschwindigkeit der Programmentwicklung.
Die mögliche zukünftige Entwicklung von ForauS sollte natürlich auch eine Änderung der Benutzungsoberfläche mit einschließen. Ansatzpunkte liegen in der weiteren Verbesserung der Manipulation von logischen Strukturelementen, der Unterstützung der vollständigen Tastatur (z.B. die Hot-Keys „Find“, „Again“, „Undo“) sowie das stabilere Zusammenspiel der einzelnen Module, insbesondere die Verbindung ForauS und daVinci. Dies setzt allerdings in den meisten Fällen eine gleichzeitige Erweiterung der anderen Module voraus.
Durch den Zeitmangel bedingt, konnten wir (gemeint ist das gesamte Projekt) einige — uns wichtige — Funktionen leider nicht verwirklichen. Hierzu zählen die Referenzen, Sprünge zu Referenzen, erweiterte Positionierungsmöglichkeiten des Cursors (siehe Menü „Properties“), die tatsächliche Unterstützung mehrerer Sichten (durch den Formatierer, der Internen Struktur und der Ausgabe), die Anzeige der logischen Elemente im Arbeitsfenster, ein verbessertes (passives oder aktives) Hilfsystem und erweiterte Blockmanipulationsmöglichkeiten. Diese Funktionen waren jedoch nicht erklärtes, primäres Ziel gewesen.
Im Wesentlichen sind deshalb die im zweiten Semester gestellten und mit einer höheren Priorität ausgezeichneten Anforderungen zu unserer Zufriedenheit erfüllt worden.
Insgesamt gesehen vermittelte die Projektzeit sehr lehrreiche Erfahrungen im Umgang mit dem Arbeiten in Teams, der Entwicklung und die Umsetzung von teilweise eigenen Ideen, dem Planen von technischen Systemen im größeren Stil und die Anwendung der im Grund- und Hauptstudium

[22] Fehler treten meist erst bei höherliegenden Funktionen auf und sind in „fremden“ Systemteilen nur schwer zu finden.
[23] Gegen Projektende wurden die Präsentationen deshalb zunehmend kürzer.

gelernten Techniken und Methoden. In jedem Fall war die Arbeit im Projekt daher sinnvoll und sollte auch weiterhin einen Teil des Informatikstudiums ausmachen.

4.8 Handbuch

Jens-Martin Brinkmann, Frank Hettling

Die Erstellung des ForauS-Textverarbeitungssystems ging in die letzten Runden. In diesem Stadium der Programmentwicklung wurde im Plenum zum Glück für zwei Studenten festgestellt, daß für dieses bahnbrechende neue Textverarbeitungssystem keinerlei Dokumentation für einen zukünftigen Benutzer verfügbar war. Deshalb entschloß sich das Projekt, eine neue Arbeitsgruppe ins Leben zu rufen. Diese Arbeitsgruppe bekam den Auftrag, ein Handbuch für das ForauS-Programm zu erstellen. Das Handbuch sollte den Benutzer in die Lage versetzen, das ForauS-Programm schnell, effizient und in seinem vollen Funktionsumfang zu beherrschen. Dieses Wissen sollte jedem Benutzer ohne fremde Hilfe und nur mit dem Handbuch gewappnet, so gut wie möglich zugänglich sein, beschloss die Handbuchgruppe in ihrem ersten Treffen. Um dieses Ergebnis zu erreichen, fing die Handbuchgruppe an, viele verschiedene Handbücher von herkömmlichen Programmen jeder Art zu untersuchen, um sie auf ihre Brauchbarkeit im Umgang mit fremden und unbekannten Programmen zu prüfen. Sie stellte mit Erschrecken fest, daß kaum ein Handbuch ihren sich selbst gestellten Anforderungen genügte. Aus den wenigen guten Handbüchern schauten die Mitglieder der Projektgruppe sich an, wie man ein gutes Handbuch in die Tat umsetzt und aus den anderen, wie man es auf keinen Fall macht (damit sich keiner auf den Schlips getreten fühlt, möchten wir hier die rein subjektive Meinung unterstreichen). Nachdem nun feststand, was zu tun war, konnte die eigentliche Arbeit beginnen. Das Programm ForauS wurde nun auf seine magere, lauffähige Funktionalität hin untersucht und die Ergebnisse mit der Gruppe Benutzungsoberfläche verglichen. Die Arbeitsgruppe Benutzungsoberfläche bestätigte die Ergebnisse. Dieses war der erste Erfolg der Handbuchgruppe. Von diesem Erfolg beflügelt, stürzten sich nun die Mitglieder der Gruppe in die Arbeit, denn der schwerste Teil lag noch vor ihnen: Die eigentliche Erstellung des Handbuches.

Am Anfang stand die Gliederung des Handbuches. Diese Gliederung wurde im Plenum präsentiert und stand zur Diskussion. In den Plenum waren nur sehr wenig Personen anwesend. Dies hatte zur Folge, daß für die nicht anwesenden Projektmitglieder die Gliederung nochmals vorgestellt werden mußte, wobei es wieder zu einer Diskussion kam. Die Ergebnisse der beiden Diskussionen waren leider unterschiedlich. Dies kam wohl daher, daß jedes Projektmitglied seine eigene Vorstellung vom Handbuch hatte und diese im Handbuch auch wiederfinden wollte. So kam es zu einer dritten Diskussion, in der nun alle Projektteilnehmer anwesend waren. Wir erhielten die endgültige Gliederung des Handbuches. Jetzt endlich konnten die Gruppenmitglieder anfangen, die Gliederung mit Leben zu füllen. Um den Inhalt des Handbuches anschaulicher zu gestalten, setzten wir in vielen Abschnitten Bilder hinzu. Mit viel Mühe und Schweiß schafften wir es, das Handbuch fertig zu stellen (Naja rein subjektiv gesehen, denn wäre das Handbuch in die Klauen eines Projektmitgliedes gefallen...).

Ein leicht mittleres Unglück zerstörte den Stolz unserer Arbeit. Dieses Unglück bezeichnet man schlechthin als Windows. Nun mußten wir wieder von vorn anfangen und waren wir in Zeitverzug geraten, denn der Abgabetermin war schon bedrohlich nahe gerückt. Aber mit guter Hoffnung und dem Wissen, um den allgemeinen Zeitverzug, erstellten wir das Handbuch unter großem Zeitdruck neu. Nach der Fertigstellung wurde das Handbuch im Plenum präsentiert. Wie immer konnte man die hohen Ansprüche einiger Projektmitglieder nicht zufriedenstellen. So mußte das

Handbuch mehrmals überarbeitet werden. Wir hoffen trotzdem, daß alle zukünftigen Benutzer des ForauS-Programms das Handbuch zu schätzen wissen.

5. Meinungen

5.1 Bernd

Hier tippe ich nun und schaue auf 2,x Jahre Projektarbeit zurück. Der Satz macht auch schon meinen ersten Kritikpunkt am Projekt deutlich: Ich hätte es besser gefunden, wenn das Projekt nach zwei Jahren beendet gewesen wäre. Dazu wären meiner Meinung zwei Dinge notwendig gewesen: Zum Einen hätten wird das Ziel unseres Projektes viel eher finden sollen (zu Beginn des zweiten Semesters), um dann mehr Zeit für die Implementierung, das Testen und das Dokumentieren zu haben. Zum anderen hätten wir die Dokumentation (ich meine den ca. 5 kg schweren Stapel gepreßten Holzes mit weißlichen Präsentationsregeln) parallel zur Implementierung schwärzen sollen. Dann wäre die Arbeit gegen Ende des vierten Projektsemesters vom Tisch und die nichtalkoholischen Getränke auf demselben gewesen.
Ich vermute, daß ein Kurs zum Thema „Projektmanagement“ uns in den o.g. Punkten deutlich geholfen hätte, die Dinge umzusetzen.
Was mich persönlich am Projekt am meisten gestört hat, ist die teilweise anzutreffende mangelnde Einhaltung von Versprechen einzelner Projektteilnehmer bezüglich der Fertigstellung ihrer Arbeiten.
Soviel zu den negativen AspecTen, die mir aufgefallen sind. Ich möchte hier jedoch manifestieren, daß die positiven Dinger derselben Sprache mir prägender in Erinnerung bleiben werden. Mir sind eigentlich kaum Spannungen zwischen Projektteilnehmern und den Projektleitern aufgefallen, ein mit Sicherheit bemerkenswerter Umstand. Meiner Meinung nach hat die Projektleitung ihre Aufgabe gut gelöst. Insgesamt empfand ich das Klima in der Gruppe als sehr gut, was bei einigen außeruniversitären Happenings auch redlich gefördert wurde (man denke an den Abend im Woodys oder das chinesische Essen ...).
Kommen wir denn zum Schluß: In den zwei Jahren der Projektdauer habe ich persönlich eine Menge Erfahrungen sammeln können, die sowohl positiver als auch negativer Natur waren. Mit dem Ergebnis des Projekts bin ich persönlich zufrieden; ich denke daß wir – wenn man die äußeren Bedingungen betrachtet – einiges auf die Beine bzw. auf den Monitor, das Papier, ... gebracht haben.

5.2 Tarik

Meine persönliche Meinung zum Projekt ForauS — gar nicht so einfach. Nun, ich denke wir alle haben eine ganze Menge gelernt, es wäre auch schlimm wenn das nicht so wäre. Wir haben vor allem gelernt, mit vielen verschiedenen Menschen zusammen zu arbeiten. Bestimmt habe ich hier am meisten profitiert.
Aber auch in Sachen Satzsystem macht uns jetzt so leicht kein Programm mehr was vor. Natürlich, es wurde auch programmiert, aber ich habe bestimmt nicht den größten Teil Programmcode geschrieben. Aber es gab ja auch noch viele andere Dinge zu tun z.B. ähhh ...

Wie dem auch sei, es hat mir wirklich *sehr* viel Spaß gemacht. Das gesamte Klima im Projekt, die vielen Feiern, auch mit den Projektleitern, waren doch echt immer lustig. Aber auch ernstere Dinge wie die gemeinsam gelösten Probleme, die schnellen und die weniger schnellen DenkerInnen und ProgrammiererInnen, die Tage vor dem Rechner. Vor allem auch unsere Diskussionsrunden bei Kaffee und Kuchen. All das wird trotz des Stresses, der uns zum Schluß gnadenlos einholte, bei mir in positiver Erinnerung bleiben.
Was ich auch sehr lobenswert finde, ist die Offenheit und die zwischenmenschlichen Kontakte zwischen den ForauSianern, die nun zum Ende des Projektes meiner Meinung nach recht stark ausgeprägt sind.
Auf jeden Fall möchte ich auch die Tagungen erwähnen, die ich als „Abgesandter“ besuchen konnte. Hier einmal über den Tellerrand zu schauen halte ich für extrem wichtig; umsomehr wundert es mich, daß so wenig Interesse bei den anderen bestand?!?
Zu guter letzt auch noch einmal vielen Dank für die super Special-Performance. Ich muß schon sagen, sie war ein voller Erfolg, oder? Da haben wir doch wirklich Teamgeist bewiesen.
Übrigens, wußtet Ihr schon:

```
----- Der Weser-Kurier berichtet:
FORAUS programmiert in ASPECT.
Tarik dazu: Gut, daß das Projekt zuende ist.
```

End Transmission...

5.3 Jan

Nach gut vier Projektsemestern ist jetzt die Zeit für einen Rückblick gekommen. Ich kann mich noch gut an das Anfangstreffen erinnern, an dem Michael und Mattias die Projektidee vorstellten. Die Overhead-Folien zeigten Bilder einer fiktiven Textverarbeitung, die eine logische Beschreibung der Textstruktur erlaubte, auf Sun Sparc Stations lauffähig war und in einer funktionalen Programmiersprache implementiert werden sollte. Im nachhinein erscheint es merkwürdig, daß ich damals die Schwierigkeit nicht in der Komplexität des Themas, sondern in der Handhabbarkeit der Programmiersprache sah (wobei ja gerade die Wahl von ASpecT noch bis ins dritte Semester emotionsgeladene Diskussionen hervorrief...).
Die ersten zwei Semester betrieben wir Grundlagenforschung, um die Anforderungen für unser System zusammenzutragen. Da kamen unter anderem jetzt so kurios anmutende Wünsche, wie Sprachnotizen, das Vorlesen bestimmter Textpassagen oder das hochgesteckte Ziel der gleichzeitigen Bearbeitung eines Dokumentes von mehreren Benutzern im Netz auf.
Im dritten Semester fiel es uns schwer, die Spezifikation für das ForauS-System aufzustellen. Die Fülle von Funktionalitäten und möglichen Lösungswegen wirkte eher verwirrend als richtungsweisend. Schließlich war es der Zeitdruck und die von Michael und Mattias aufgestellten *Meilensteine*, die uns zu Entwurfsentscheidungen zwangen.
Das vierte und letzte Semester stand dann ganz im Zeichen der Implementierung; ich war sehr überrascht, wie schnell es dann schließlich gelang, ein lauffähiges Gesamtsystem zu entwickeln, in dem unsere Grundkonzepte der oben skizzierten Projektidee tatsächlich verwirklicht worden waren.
Für mich war dieses Projekt, neben dem Software-Praktikum, die erste größere Programmiertätigkeit, die nicht mehr alleine, sondern in der Gruppe mit mehreren Beteiligten gelöst wurde. Das Geheimnisprinzip, die Black-Box-Programmierung und die Kommunikation über vorher fest definierte Schnittstellen lernte ich hier in der Praxis kennen. Doch nicht nur der Umfang und die

Methoden, sondern auch die Komplexität hatte neue Ausmaße. Welch ein Triumph war es, als wir in unserer Gruppe eine doppelt verschränkte Rekursion über den Baum der Formatiererstruktur durch ein mehrzeiliges `map`-Konstrukt gelöst hatten!
Das Klima und die Zusammenarbeit in unserem Projekt fand ich im allgemeinen gut. Dabei hat mir die eher im Hintergrund verbleibende Projektleitung durch Michael und Mattias gut gefallen. Auch wenn zu bemerken wäre, daß durch früheres Einlenken in der Phase der Orientierungslosigkeit des dritten Semesters vielleicht ein rechtzeitigeres Projektende hätte erreicht werden können, war dieser Verlauf doch eher in der eigentlichen Projektintention des selbständigen, nach wissenschaftlichen Maßstäben vorgehenden, Erarbeitens eines Projektziels. Dabei sind wir meiner Meinung nach mit derart geringem Vorwissen den Richtlinien der Softwaretechnik und des Projektmanagements recht nahe gekommen.
Die menschlichen Höhepunkte sind für mich die Projektwochenenden in Syke und unsere außeruniversitären Aktivitäten gewesen. Die verrauchten Abende im Kaminzimmer der Jugendherberge, das chinesisch-vietnamesiche Essen bei Gunnar in Osterholz, der Abend im Woodies nach gemeinsamen Grillen im Garten und unsere „Kohl und Pinkel"-Tour sind nur einige Eindrücke, die ich sicherlich nicht vergessen werde. Aber bevor ich jetzt endgültig sentimental werden, bleibt ja noch tröstend der Ausblick auf unsere Projektabschlußfeier und die Zuversicht, daß wir ein Textverarbeitungssystem geschaffen haben, das die Welt verändern wird. In diesem Sinne...
Mit ForauS ins nächste Jahrtausend...

5.4 Jens „P."

Das Projekt war ausgezeichnet. In den vier Semestern habe ich viel über Teamarbeit mit all ihren Vorteilen (Arbeit delegieren) und Nachteilen (delegierte Arbeit nicht gemacht worden) erfahren. Ich habe gelernt, in Diskussionen die eigene Meinung zu behaupten aber auch Kompromisse zu finden. D.h. alle gruppendynamischen Aspekte waren äußerst lehrreich. Dagegen waren die formalen Inhalte des Projektes, das Spezifizieren und Ausprogrammieren einer Aufgabe, nicht beispielhaft, wie ich es vorher erwartet hätte. Dem Projekt fehlte ein Projektmanagement, das von Anfang an die vorgegebenen Ziele an Termine binden würde. In unserem Projekt erfolgt dies erst im dritten Semester mit einer merklichen Effizienzsteigerung. Es fanden keine softwaretechnischen Überlegungen statt. Wir programmierten ohne Konzept. Beides hätte von der Projektleitung forciert werden müssen. Doch diese Erfahrungen nehme ich mit in mein nächstes Projekt, wo immer es stattfinden mag. Und spätestens dann werde ich mich wieder dieses Projektes mit seinen Teilnehmern und Leitern erinnern. Live long and prosper.

5.5 Michael

Etwas mehr als zwei Jahre ist es nun her, als es für mich — und alle anderen meines „Jahrgangs" — eine schwere Entscheidung zu treffen galt: welchem der angebotenen Projekte will ich die nächsten zwei Jahre meines Lebens opfern? Die Wahl war nicht einfach, hätte ich doch eigentlich lieber etwas ganz anderes gemacht als das, was ich dort im Angebot fand. Doch jammern half nicht, das Semester begann und eine Entscheidung mußte her. Und so wählte ich, für den geneigten Leser unschwer zu erraten, ForauS.
Daß dies nicht die schlechteste Idee war, stand für mich spätestens nach dem ersten Projektsemester fest. Hier gelang es uns tatsächlich, ein bald hundert Seiten kräftiges Dokument zu formulieren, welches die „Anforderungen an ein innovatives Dokumentenverarbeitungssystem" beschreibt. Im anschließenden zweiten Semester extrahierten wir dann die Passagen aus dem Dokument, die

in der späteren Implementierungsphase realisiert werden sollten. Mit anderen Worten: das Projekt ruhte sich ein wenig auf seinen Lorbeeren aus. So kam es dann auch zu einer der rar gesähten Situationen, in denen die Betreuer durch die Einführung von „Meilensteinen“ steuernd in den Ablauf des Projekts eingriffen. Spätestens aber mit Beginn der von vielen langersehnten Programmierung wurde dies wettgemacht. Und wie unglaublich gut war das Gefühl, als zum ersten mal formatierter Text auf dem Bildschirm der Workstation aufleuchtete!
Blickt man vom heutigen Standpunkt aus einmal zurück, so kann man feststellen, daß sich die ForauSianer durchgehend sehr gut miteinander verstanden. Nicht daß man sich immer gleich in die Arme gefallen ist, die eine oder andere Meinungsverschiedenheit gab es schon mal, aber das ist, so denke ich, wohl eher normal und gehört dazu. Wie hätten auch sonst diese wunderbar knackigen Diskussionen entstehen können?
Das wirklich besondere an ForauS war wohl, daß, vor allem verglichen mit anderen parallel laufenden Projekten, fast alles von den Teilnehmenden selbständig erarbeitet wurde. Eine echte Herausforderung, denn niemand schrieb einem fest vor, was zu tun und was zu lassen war. Dies kann man sicherlich auch am Endprodukt erkennen, welches wahrscheinlich nicht unbedingt dem entspricht, was man sich in der Projektleitung ursprünglich mal gedacht hatte. Trotzdem, oder gerade deshalb, sind alle mit dem Ergebnis zufrieden. Rundherum also ein gelungenes Projekt, wie ich es jedem bremer Studenten der Informatik nur wünschen kann. Und wer weiß, vielleicht heißt es ja auch bald schon:

ForauS— erhältlich im gut sortierten Fachhandel!

5.6 Guido

Ich war anfangs gar nicht begeistert. TI-Thema bei Kreo, Editor bei Fröhlich, Compilerbau bei Gersdorf oder Bürgerinfo bei Kubicek. Ein Produktionsprojekt bei Bruns oder Rödiger haben wir nur fast auf die Beine gestellt. Da schien Bürgerinfo das geringste Übel zu sein. Nach dem ersten Treffen mußte ich[1] meine Meinung etwas korrigieren. Trotz der gruseligen Vorstellung des ForauS-Projektes schaute ich mir danach ein Projekttreffen mit an. Irgend eine „Wahrheit über ForauS“ sollte verkündet werden. „Wer hat vor, in dem Projekt zu bleiben?“ - „Mal sehen“. Aber genau das ergab sich.
Eine wichtige Erfahrung bei den ersten Ausarbeitungen für mich war das Gefühl, daß unsere Vorträge für die anderen Projektmitglieder waren. Das erste Mal stellte man etwas vor, das nicht primär dem Erwerb eines Scheines diente. Es war unser Projekt. Dieses Gefühl wurde natürlich auch dadurch möglich, da die Projektleitung nur wenig steuernd (im Sinne von vorgebend) und restriktiv eingriff. Wir konnten bestimmen, was wir tun wollten. Hierfür ein dickes Lob in Richtung Michael und Mattias.
Auf der anderen Seite bestand dadurch die Gefahr, daß das Projekt sich festfährt oder es versäumt, auf gewisse Schwerpunkte zu achten. Hiermit meine ich vor allem die nicht konsequente Abstimmung des Ablaufes bei der Software-Erstellung; auf Konzepte des Software-Engineering wurde zu wenig Augenmerk gelegt. Etwas spät kamen die von der Projektführung aufgestellten „Meilensteine“, als sich das Projekt tatsächlich festgefahren hatte, aber sie kamen und brachten wieder Schwung in die Projektarbeit. „Aus Fehlern lernt man“. Es ist nicht wichtig, daß alles perfekt läuft, im Gegenteil. Gerade diese Dinge sind enorm wichtig.
Wenn einige Projektmitglieder aufgrund mangelnden Interesses, mangelnder Zeit oder sonstiger

[1] Ich schreibe hier in der Ich-Form. Tatsächlich war zu jedem Zeitpunkt aber auch Tarik mit involviert.

Gründe nicht die von ihnen erwarteten Leistungen erbringen[2], hat dies zwei Seiten. Einerseits, das muß ich mir selber eingestehen, kann man seine eigene Position dadurch in einem besseren Licht sehen (der Einäugige ist der König unter den Blinden), andererseits schadet es aber dem Projektfortkommen erheblich. Und genau das ist der Grund, warum ich von einigen etwas enttäuscht bin. Namen werde ich in diesem Zusammenhang aber nicht nennen (im Gegensatz zu anderen Leuten in diesem Bericht).
Aber es gab auch solche, die das Projekt enorm vorangebracht haben. Vor allem Achim ist es meiner Meinung nach zu verdanken, daß das Projekt so gut verlaufen ist. Das Klima in dem Projekt war eines der angenehmen Dinge. Es fiel mir leicht, mich mit dem Projekt zu identifizieren, obwohl ein Freund vorwarf, wir würden das Rad neu erfinden.
Das Programm ForauS, was letztendlich implementiert worden war, war eigentlich sekundär. Ich muß aber gestehen, daß mich der Fortgang der Implementierung am Ende doch angenehm überrascht hat.
Der absolute Höhepunkt des Projektes war aber die Special-Performance am Projekttag. Die Arbeit daran und die Kreativität aller Beteiligten, die hier bewiesen wurde, war einfach Spitze. Eine solche Vorführung in einer relativ kurzen Zeit auf die Beine zu stellen, und das von Informatikern! Aus diesem Grund hier nun der letzte Akt von „ForauS-Trek“:

Data: Captain, die Kreolaner haben sich von ihren Bildschirmen zurückgezogen.

Riker: Auch auf Bi-S 9 sind keine Bürger mehr zu registrieren. Es war wohl zu langweilig.

Picard: Auch wir sind nur noch wenige Lichttage von unserem Raumdock entfernt.

Troi: Ich spüre bei der Besatzung Erleichterung, aber auch ein wenig Traurigkeit.

Data: Wurden die Personalbewertungen denn schon ausgegeben?

Picard: Wollen wir hoffen, daß der Schein uns nicht trübt, äh trügt. Die BKBorgs sind nicht zu berechnen.

Troi: Captain, wie wird es weitergehen, jetzt wo die Mission beendet ist?

Picard: Deanna. Der Weltraum ist so groß und es gibt noch so vieles, von dem wir keine Ahnung und keine Vorstellung haben.
Die Besatzung wird auseinandergehen. Das Erlebte aber wird uns für alle Ewigkeit weiter verbinden.
Nun wird es aber Zeit. Mr. LaForge! Direkter Kurs in den Diplom-Sektor,
Warp 9,6. Energie!

[2] Das Projekt muß wegen der enormen Bedeutung für das Diplom eine führende Rolle in der persönlichen Prioritäteneinschätzung einnehmen.

Literaturverzeichnis

[AFQ88] André, Furuta, and Quint. Interactively editing structured documents. *Electronic Publishing: Origination, Dissemination and Design*, 1(1):19–44, April 1988.

[AFQ89] André, Furuta, and Quint. *Structured Documents.* Cambridge University Press, 1989.

[App87] Apple Computer Inc. *Human Interface Guidelines: The Apple Desktop Interface.* Addison Wesley, New York, 1987.

[BB90] A. Brown Jr. and H. Blair. A logic grammar for document representation and document layout. In R. Furuta, editor, *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, Cambridge, New York, 1990. Cambridge University Press.

[BK89] A. Brüggemann-Klein. *Einführung in die Dokumentenverarbeitung.* B.G. Teubner, Stuttgart, 1989.

[DIN88] Deutsches Institut für Normung. *Bildschirmarbeitsplätze, Grundsatz ergonomischer Dialoggestaltung*, 1988. DIN 66 234 Teil 8.

[DIN94] Deutsches Institut für Normung. *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten*, 1994. DIN EN 29 241 Teil 10 (ISO/DIS 9241-10).

[Fac94] Fachgruppe 4.9.2 Multimediale elektronische Dokumente der Gesellschaft für Informatik e.V. (GI). *SGML in der Praxis*, Europäisches Zentrum für Netzwerkforschung (ENC) in Heidelberg, April 1994.

[For93] Projekt Foraus. Anforderung an ein innovatives Dokumentenverarbeitungssystem. Entstand im Laufe des Projektes, 1993.

[Fra93] Frame Technology Corp. *Framemaker 4 Benutzungshandbuch.* San Jose, USA, 1993.

[Fur89] R. Furuta. Concepts and models for structured documents. In *Structured Documents.* Cambridge University Press, 1989.

[FW95] M. Fröhlich and M. Werner. Demonstration of the interactive graph visualization system davinci. In R. Tamassia and I. Tollis, editors, *Proceedings GD '94 LNCS No. 894.* Springer-Verlag, 1995.

[GJ89] D. Gierlach and R. Jankowski. Operationalisierung der Grundsätze ergonomischer Dialoggestaltung nach DIN 66 234 Teil 8 – Beispiel Textverarbeitung. *Angewandte Informatik*, 1(5):189–196, May 1989.

[Gol90] C. F. Goldfarb. *The SGML Handbook.* Oxford University Press, New York, 1990.

[Isl92] Island Graphics Corporation, San Rafael, USA. *Island Presents Demo CD-ROM*, 1992.

[ISO91] International Organization for Standardization. *Information technology – Text and office systems – Document Style Semantics and Specification Language (DSSSL)*, 1991. ISO/DEC DIS 10179.

[Job89] V. Jobloboff. Document representation. In *Structured Documents*. Cambridge University Press, Cambridge, New York, 1989.

[Knu86] D. Knuth. *The TEX book*. Addison Wesley, Reading, MA u.a., 1986.

[Kop89] H. Kopka. *LATEX: Eine Einführung*. Addison Wesley, Bonn; München; Reading, MA u.a., zweite edition, 1989.

[KTR84] H. Klocke, S. Trispel, and G. Rau. Entwicklung einer Mensch-Rechner Schnittstelle für ein Anästhesie-Informationssystem unter Berücksichtigung ergonomischer Gesichtspunkte. *Angewandte Informatik*, 1(5):197ff, May 1984.

[Lam86] L. Lamport. *LATEX: A Document Preparation System*. Addison Wesley, Reading, MA, 1986.

[Mey90] B. Meyer. *Objektorientierte Softwareentwicklung*. Hanser-Verlag, München, 1990.

[Opp89] R. Oppermann. Gestaltung der Mensch-Maschine-Kommunikation. *Informationstechnik*, 1(3):181–189, March 1989.

[PF93] Arbeitsgruppe Präsentationsregeln Projekt Foraus. Foraus: Die präsentationsregeln. Entstand im Laufe des Projektes, 1993.

[QV86] V. Quint and I. Vatton. GRIF: An interactive system for structured document manipulation. In J.C. van Vliet, editor, *Text Processing and Document Manipulation*, pages 200–213. Cambridge University Press, April 1986.

[Sun90] Sun Microsystems Inc. *OPEN LOOK Graphical User Interface Application Style Guidelines*. Addison-Wesley, New York, 1990.

[vHS93] J. von Holten and R. Seifert. *ASpect 2.0 User Manual*. Universität Bremen, 1993.