Helmut Jarosch

Information Retrieval und Künstliche Intelligenz

DUV

Helmut Jarosch

Information Retrieval und Künstliche Intelligenz

WIRTSCHAFTSINFORMATIK

Helmut Jarosch

Information Retrieval und Künstliche Intelligenz

Deutscher Universitäts-Verlag

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
<http://dnb.d-nb.de> abrufbar.

1. Auflage April 2007

Lektorat: Brigitte Siegel / Britta Göhrisch-Radmacher

Der Deutsche Universitäts-Verlag ist ein Unternehmen von Springer Science+Business Media.
www.duv.de

Umschlaggestaltung: Regine Zimmer, Dipl.-Designerin, Frankfurt/Main
Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier
Printed in Germany

ISBN 978-3-8350-0598-3

Vorwort

In nahezu allen Unternehmen gilt heute die effiziente Nutzung der Ressource „Wissen“ als einer der kritischen Erfolgsfaktoren; dem ***Wissensmanagement*** wird immer größere Aufmerksamkeit geschenkt. Häufig steht man dabei vor dem Problem, aus einer ständig wachsenden Menge gespeicherter Text-Dokumente die für eine aktuelle Fragestellung relevanten Dokumente herauszusuchen.

Information-Retrieval-Systeme sollen den Benutzer bei der Informationssuche unterstützen. Die verfügbaren Information-Retrieval-Systeme bieten dem ungeübten Benutzer jedoch zu wenig Unterstützung. Da die zu recherchierenden Texte in der Regel ohne lexikalische Kontrolle – und häufig noch dazu in verschiedenen Sprachen - verfasst wurden, ist es dem Benutzer kaum möglich, die treffenden Termini für seine Suchanfrage zu „erraten“. Die Benutzeroberfläche solcher Systeme muss so gestaltet werden, dass auch der ungeübte Benutzer seinen Informationsbedarf in einfacher Weise formulieren kann.

Nach der im vorliegenden Buch entwickelten Methode soll dieses Ziel erreicht werden, indem Methoden, die im Forschungsgebiet der ***Künstlichen Intelligenz*** (KI) entwickelt wurden, auf das Problem des Information Retrieval übertragen werden: Zwischen den Benutzer und das Information-Retrieval-System soll sich ein sogenannter ***KI-Assistent*** schieben, der dem Benutzer in dreierlei Hinsicht Hilfe anbietet:

1. Durch eine geeignete Menüführung soll der Benutzer dabei unterstützt werden, seine Suchanfrage nach einer semantisch orientierten Methode zu konstruieren.

2. Mittels statistischer Verfahren sollen aus den textorientierten Recherche-Ergebnissen Wörter ausgewählt und dem Benutzer zur Aufnahme in seine Begriffskonstrukte angeboten werden.

3. Unter Verwendung von Methoden des Begriffslernens soll der Benutzer dazu angeleitet werden, durch Boole'sche Operatoren und Kontext-Operatoren eine geeignete Verknüpfung seiner Begriffskonstrukte vorzunehmen, so dass akzeptable Recherche-Ergebnisse erzielt werden.

Um diese Ziele schnell und effizient erreichen zu können, wird zunächst eine spezielle ***System-Architektur*** entwickelt, durch die keine Eingriffe in die Software des Information-Retrieval-Systems erforderlich sind. Ein KI-Assistent, der zusätzliche Intelligenz in den Rechercheprozess einbringen soll, vermittelt zwischen dem Benutzer und dem Information-Retrieval-System und hebt dadurch die Benutzerkommunikation auf ein höheres Niveau.

Die Kommunikation des KI-Assistenten mit dem Benutzer und mit dem Information-Retrieval-System sowie die Steuerungs- und Nachrichtenflüsse, die zwischen den Komponenten des KI-Assistenten ablaufen, werden durch ein ***Modell abstrakter Maschinen*** beschrieben.

Für die Formulierung der Algorithmen, nach denen die abstrakten Maschinen arbeiten sollen, wird die ***Programmiersprache*** KOMPROMISS entworfen, die eine objektorientierte Programmierung ermöglicht.

Um die Leistungsfähigkeit dieses Ansatzes zu demonstrieren, wird ein spezieller ***KI-Assistent*** entwickelt, der den Benutzer bei der Online-Recherche unterstützen soll. Die dafür eingesetzten Lernverfahren aus dem Gebiet der Künstlichen Intelligenz und die Algorithmen der Massendatenverarbeitung werden durch Teilsysteme abstrakter Maschinen realisiert. Ihre objektorientierte Architektur und ihre Kommunikationsflüsse werden aus der Aufgabenstellung abgeleitet. Das Niveau der Software ist dabei durch abstrakte Datentypen und Koprogramme charakterisiert.

Wichtige Anregungen für die Konzeption dieses Buches gehen auf Diskussionen mit Herrn Prof. Dr. Erich Mater und Herrn Prof. Dr. Fritz Wysotzki zurück. Ihnen sei an dieser Stelle herzlich gedankt. Dank gilt auch der Fachhochschule für Wirtschaft Berlin, die mir die ungestörte Arbeit an diesem Buch ermöglichte.

Helmut Jarosch

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CICS	***C***ustomer ***I***nformation ***C***ontrol ***S***ystem
DBMS	***D***ata ***B***ase ***M***anagement ***S***ystem (***D***aten***B***ank-***M***anagement***S***ystem)
DSO	***D***ata ***S***tar ***O***nline
IRS	***I***nformation-***R***etrieval-***S***ystem
KI	***K***ünstliche ***I***ntelligenz
KOMPROMISS	***KOM***munikations- und ***PRO***zess***M***onitor für ***I***nformations-***S***y***S***teme
LIFO	***L***ast ***I***n ***F***irst ***O***ut
OS	***O***peration ***S***ystem
SQL	***S***tructured ***Q***uery ***L***anguage
STAIRS	***ST***orage ***A***nd ***I***nformation ***R***etrieval ***S***ystem
UML	***U***nified ***M***odeling ***L***anguage
VBA	***V***isual ***B***asic for ***A***pplications

Symbolverzeichnis

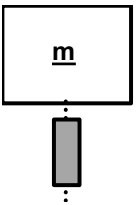
Allgemeine Schreibweisen

Aktivator	Fachbegriff, der im Buch definiert wird (wird beim ersten Auftreten ***fett/kursiv*** gesetzt)
'MASCHINE'	Bezeichnung für eine abstrakte Maschine oder für ein Teilsystem abstrakter Maschinen
'Algorithmus'	Bezeichnung für einen Algorithmus
`MESSAGE`	Sprachelement der Programmiersprache KOMPROMISS
Z^+	Plushülle der Menge Z
Z^*	Sternhülle der Menge Z
$\otimes$	undefinierter Wert
$\circ$	Ausführung einer Operation

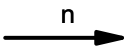
Notationen für die Grammatik-Beschreibung

<Zahl>	Metasymbol
"x"	Terminalsymbol x
::=	Definitionszeichen
|	Trennzeichen für Alternativen
$[\ldots]_a^b$	a- bis b-malige Wiederholung einer Konstruktion

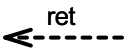
Notationen in UML-Diagrammen



abstrakte Maschine (UML: Objekt) m im Sequenzdiagramm mit Lebenslinie und Aktivierungsbalken

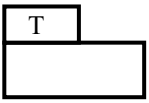


Aktivierung (UML: Aufruf) einer abstrakten Maschine durch Zusenden der Nachricht n

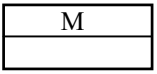


Rücksendung der Antwort ret an den Aktivator

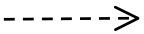
Ende der Existenz einer abstrakten Maschine (**X**)



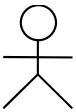
Teilsystem (UML: Subsystem) T abstrakter Maschinen im Paketdiagramm



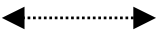
abstrakte Maschine (UML: Objekt) M im Paketdiagramm



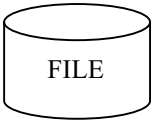
Generierung (UML: Instanziierung) einer abstrakten Maschine



Benutzer (UML: Akteur) des Information-Retrieval-Systems



Kommunikation (UML: Assoziation) des KI-Assistenten mit dem Benutzer



externe Datei (UML: Datenspeicher)

Symbole in den Formeln

A^t	Menge der zum Zeitpunkt t im Wörterbuch gespeicherten Wortformen, die aus den Begriffskonstrukten stammen
B^t	Menge der zum Zeitpunkt t im Wörterbuch gespeicherten Wortformen, die aus relevanzbewerteten Dokumenten stammen
C	übereinstimmende Struktur aller Wörter, die zu einer Wortklasse gehören
D	Alphabet der Ziffern
E	Menge von elementaren Endungen
$E(\tilde{p})$	Funktion zur Ermittlung der Anzahl der Selektionsoperationen, die bei einer Reduktion des Klassifikators $K_{\tilde{p}}$ eingespart werden
$F\left(w, W^t\right)$	Funktion zum Aufsuchen der Wortform w in der Wortmenge W^t
G	Grammatik
$G\left(K_{\tilde{q}_1}, K_{\tilde{q}_2}\right)$	Funktion zur Ermittlung der Anzahl der Selektionsoperationen, die durch eine Überlagerung der beiden Klassifikatoren $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ eingespart werden
H	absolute Häufigkeit
H_0	Hypothese
$I(m)$	Identifikator der abstrakten Maschine m
$K_{\tilde{p}}^{\nu}$	Klassifikator für die Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$

L	Alphabet der Buchstaben
M^t	Menge der abstrakten Maschinen zum Zeitpunkt t
N_x	Anzahl, die sich auf die gesamte Datenbank bezieht
Q^t	Menge der zum Zeitpunkt t im Wörterbuch gespeicherten Wortformen, die dem Benutzer ***nicht*** angeboten werden sollen
R	Menge der Begriffskonstrukte in der Suchanfrage
S	Alphabet der Sonderzeichen
S_r	Suchanfrage zum Begriffskonstrukt r
S_0	Grobanfrage als erste Stufe des Klassifikators
T	abstrakter Datentyp
$T\left(w¤n, W^t\right)$	Funktion zur Suche nach dem Truncation-Wort $w¤n$ in der Wortmenge W^t
U^t	Menge der zum Zeitpunkt t im Wörterbuch gespeicherten Wortformen, die aus unbewerteten Dokumenten stammen
$Ü_v\left(K_{\tilde{q}_1}, K_{\tilde{q}_2}\right)$	Überlagerung der beiden Klassifikatoren $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$
V_π	Menge der Attribute zum Präfix π
W	Wortmenge
W^t	Menge der zum Zeitpunkt t im Wörterbuch gespeicherten Wortformen
Z	Alphabet von Symbolen

$\underline{\text{act}}(\text{I})$	Empfängerangabe beim Aktivieren der abstrakten Maschine mit dem Identifikator I
$\underline{\text{act}^+}(\text{I})$	Empfängerangabe im Endpunkt zur Weitergabe der Steuerung an die abstrakte Maschine mit dem Identifikator I
$\underline{\text{aret}}$	Empfängerangabe im Aktivator-Rückkehrpunkt
$\underline{\text{aret}^+}$	Empfängerangabe im Endpunkt zur Weitergabe der Steuerung an den Aktivator
b	Bewertungsmaß einer Wortform
c	Empfängerangabe im Sendepunkt
d	Datenobjekt einer abstrakten Maschine
e	Endung bzw. Endungsfolge
$\underline{\text{exec}}(\mathcal{A})$	Empfängerangabe beim Aufruf einer abstrakten Maschine vom Typ „Routine“, die nach dem Algorithmus $\mathcal{A}$ arbeitet
$\underline{\text{gen}}(\mathcal{A},\text{I})$	Empfängerangabe beim Generieren einer neuen abstrakten Maschine, die nach dem Algorithmus $\mathcal{A}$ arbeitet und deren Identifikator der Variablen I zugewiesen wird.
$\underline{\text{gret}^+}$	Empfängerangabe im Endpunkt zur Weitergabe der Steuerung an den Generator
h	relative Häufigkeit
m	abstrakte Maschine
m_a	Aktivator
m_g	Generator
m_0	Monitormaschine

n	Nachricht
n_x	Anzahl, die sich auf eine Teilmenge der Datenbank bezieht
p	Wahrscheinlichkeit
$\tilde{p}$	Folge von Selektionsoperationen
q	Quantil einer statistischen Prüfverteilung
$\tilde{q}_i$	spezielle Folge von Selektionsoperationen
r	Begriffskonstrukt
s	normierte Wortform
s_a	Aktivierungspunkt
s_{ar}	Aktivator-Rückkehrpunkt
$s_{\mathcal{A}}$	Sendepunkt des Algorithmus $\mathcal{A}$
$s^+_{\mathcal{A}}$	Endpunkt des Algorithmus $\mathcal{A}$
s_g	Generierungspunkt
s_{ua}	Unterprogramm-Aufrufpunkt
t	Zeitpunkt
w	Wortform
x	eine der beiden Klassen „ausgeben“ oder „nicht ausgeben“
y	zu analysierender Rest eines Textes
z	Symbol aus einer Wortform
z_{Σ_i}	Zweckmäßigkeitsmaß der Selektionsoperation Σ_i

$\mathcal{A}$	Algorithmus
$\mathcal{F}_{\tilde{p}}^{\nu}$	Objektmenge von Fundorten auf dem Niveau ν, die durch die Folge von Selektionsoperationen $\tilde{p}$ gewonnen wurde
$\mathcal{M}^t$	System der abstrakten Maschinen zum Zeitpunkt t
$\mathcal{N}$	Menge der Dokumente, die nicht ausgegeben werden sollen
$\mathcal{P}^{\nu'}(\varphi)$	Funktion zur Projektion der Fundortbeschreibung φ auf den ν'-dimensionalen Raum
$\mathcal{W}^t$	Wörterbuch zum Zeitpunkt t
$f_{\tilde{p}}^{\nu}$	Lernmenge von Fundorten auf dem Niveau ν, die durch die Folge von Selektionsschritten $\tilde{p}$ gewonnen wurde
$s(\beta_1, \beta_2)$	Funktion zur Kumulierung der beiden Bewertungsmaße β_1 und β_2
t	signumbehaftetes Begriffskonstrukt
α	Irrtumswahrscheinlichkeit
α^t	Aktivierungsrelation zum Zeitpunkt t
β	kumuliertes Bewertungsmaß einer Wortform
β_i	kumuliertes Bewertungsmaß einer Wortform hinsichtlich der irrelevanten Dokumente
β_r	kumuliertes Bewertungsmaß einer Wortform hinsichtlich der relevanten Dokumente
Γ	System der Begriffskonstrukte

γ^t	Generierungsrelation zum Zeitpunkt t
ε	leeres Wort
η	Markierung des aktuellen Morphems
κ	Koordinate eines Fundortes
κ_μ	Morphemklasse des Morphems μ
λ	Lesekopf
μ	Morphem
ν	Niveau einer Fundortbeschreibung
ξ	Signum einer Selektionsoperation
π	Präfix einer Wortform
ρ	Suffix einer Wortform
ρ_R	Relation der Referenzen über der Menge der Begriffskonstrukte R
Σ	Menge von Selektionsoperationen
σ	Selektionsoperation
$\tau_{\tilde{p}}$	Menge von signumbehafteten Begriffskonstrukten
$\Phi(w)$	Funktion zur Prüfung, ob die Symbolfolge w eine Endungsfolge ist
φ	Fundortbeschreibung
χ^2	Verteilungsfunktion
Ψ_C	Präfixbaum zur Speicherung von Wortformen mit der gemeinsamen Struktur C

1 Einführung

1.1 Wissen als kritischer Erfolgsfaktor

Als einer der kritischen Erfolgsfaktoren gilt heute in den Unternehmen die effiziente Nutzung der Ressource „Wissen“. Deshalb wird dem Wissensmanagement immer größere Aufmerksamkeit geschenkt.[1] Moderne Verfahren des ***Information Retrieval*** haben in den letzten Jahren an Bedeutung gewonnen.[2] Waren diese Verfahren lange Zeit nur auf spezialisierte bibliographische Informationssysteme beschränkt, so erlangten sie durch das Aufkommen von ***Suchmaschinen*** im World Wide Web mittlerweile eine zentrale Stellung in der Informationsversorgung. Der Suchmaschinen-Markt zeichnet sich durch eine hohe Konzentration aus.[3] Die drei wichtigsten Suchmaschinen sind heute:

- Google: Die Suchmaschine erreicht die höchsten Nutzerzahlen.
- Yahoo: Der Host hat ein Portalangebot, in dem die Web-Suche integraler Bestandteil ist.
- Microsoft: Die Suchdienste werden unter der Bezeichnung MSN (Microsoft Network) betrieben.

1.2 Aufgaben des Information-Retrieval-Systems

Der Zugang zur wissenschaftlich-technischen Information erfolgt heute fast ausschließlich im Dialog durch die Nutzung automatisierter ***Information-Retrieval-Systeme (IRS)***. Dabei ist eine Situation eingetreten, bei der eine ständig wachsende Anzahl von Personen mit diesen Systemen in Kommunikation tritt. Von

---

[1] Vgl. beispielsweise Stock (2000) und Hölscher (2002).
[2] Vgl. beispielhaft Fuhr (2004) bzw. Lewandowski (2005).
[3] Die Entwicklungen auf dem Suchmaschinen-Markt werden ausführlich in Karzauninkat (2003) dargestellt, vgl. auch Ojala (2002).

einer niveauvollen Abwicklung dieser Kommunikation hängt in starkem Maße die effiziente Informationsversorgung der Benutzer ab.

Für die Benutzerkommunikation stellen die verschiedenen Informationsdienste ***Retrieval-Sprachen*** bereit. Prominente Vertreter dieser Sprachen sind:

- DSO (Data Star Online) für die Informationssuche in den Datenbanken des Hosts DIALOG DATA STAR.
- DIALOG für die umfangreichen Retrieval-Möglichkeiten, die der Host DIALOG bereitstellt.
- MESSENGER für die Retrieval-Funktionen des Hosts STN International.

Ein Information-Retrieval-System muss zwei Hauptfunktionen erfüllen:

1. Es muss einer Vielzahl von Benutzern den stabilen Fernzugriff zu bibliographischen Datenbanken gewähren. Dazu gehört die Lösung solcher Aufgaben wie beispielsweise die Speicherung großer Dokumentenmengen, das effiziente Retrieval, die Sicherung des Multi-user-Betriebs und die Gewährleistung des Zugriffsschutzes. Diese Aufgaben werden von den verfügbaren Information-Retrieval-Systemen zufriedenstellend gelöst.

2. Es muss jedem Benutzer die Möglichkeit bieten, ohne spezielle Kenntnis der Techniken des Information Retrieval qualitativ hochwertige Suchergebnisse zu erzielen. Die heute verfügbaren Information-Retrieval-Systeme geben jedoch dem gelegentlichen Benutzer zu wenig Unterstützung.

Der Schwachpunkt der verfügbaren Information-Retrieval-Systeme liegt somit in der mangelnden Unterstützung der Benutzer bei der Formulierung ihrer Suchanfrage. Da die zu recherchierenden Texte in der Regel ohne lexikalische Kontrolle – und häufig noch dazu in verschiedenen Sprachen - verfasst wurden, ist es dem Benutzer kaum möglich, die treffenden Termini für seine Suchanfrage zu „erraten“.

Die Benutzeroberfläche von Information-Retrieval-Systemen sollte nach Methoden gestaltet werden, die es auch dem ungeübten Informationssuchenden ermöglichen, seinen Informationsbedarf in einfacher Weise zu formulieren. Das lässt

sich durch die Übertragung von Methoden der ***Künstlichen Intelligenz*** (KI) auf das Problem des Information Retrieval erreichen. Das Ziel ist es, zusätzliche Intelligenz in die Kommunikation mit dem Information-Retrieval-System einzubringen, um die Mensch-Maschine-Kommunikation auf ein qualitativ höheres Niveau zu heben. Dazu muss Software geschaffen werden, die einerseits mit dem IRS einen programmierten Dialog in der Kommandosprache führt, andererseits die vom IRS erhaltenen Informationen problemspezifisch auswertet und dabei in besonderen Situationen in Kommunikation mit dem Benutzer tritt. Der Dialog mit dem Benutzer ist nun nicht mehr an die Ebene der Kommandosprache gebunden, sondern kann auf einem qualitativ höheren Niveau geführt werden, das der jeweils zu lösenden Aufgabe besser angepasst ist.

1.3 Aufgaben des KI-Assistenten

Zwischen den Benutzer und das Information-Retrieval-System soll sich eine zusätzliche Software-Komponente schieben, die unter Verwendung von Methoden der Künstlichen Intelligenz den Benutzer bei der Formulierung seiner Suchanfrage unterstützt und die wir deshalb als ***KI-Assistent*** bezeichnen. Sie bietet dem Benutzer folgende Hilfen an:

- Durch eine geeignete Menüführung wird der Benutzer dazu angehalten, seine Suchanfrage nach einer semantisch orientierten Methode zu konstruieren.

- Mittels statistischer Verfahren werden Wörter aus den textorientierten Recherche-Ergebnissen ausgewählt und dem Benutzer zur Aufnahme in seine Begriffskonstrukte angeboten.

- Unter Verwendung von Methoden des Begriffslernens wird der Benutzer dazu angeleitet, durch Boole'sche Operatoren und Kontext-Operatoren eine geeignete Verknüpfung seiner Begriffskonstrukte vorzunehmen, so dass akzeptable Recherche-Ergebnisse erzielt werden.

Das vorliegende Buch befasst sich mit der Aufgabe, die Kommunikation des Benutzers mit dem Information-Retrieval-System durch einen KI-Assistenten zu unterstützen. Dazu soll zunächst ein theoretisches Modell entwickelt werden, das die Steuerungs- und Kommunikationsflüsse im Zusammenspiel von IRS und KI-Assistent als ein zeitabhängiges System abstrakter Maschinen beschreibt. Auf

der Grundlage dieses Modells soll dann eine Programmiersprache entworfen und implementiert werden, die die Programmierung des KI-Assistenten ermöglicht. Zur Demonstration der Leistungsfähigkeit dieser Sprache soll auf dem Datenbankrechner ein KI-Assistent zur Unterstützung der Online-Recherche installiert werden. Dabei sollen Lernalgorithmen und Algorithmen der Massendatenverarbeitung Anwendung finden.

2 Entwurfsentscheidungen

2.1 Erweiterungen auf dem Client oder auf dem Server?

Will man durch einen KI-Assistenten zusätzliche Intelligenz in die Kommunikation mit einem IRS einbringen, so sind mehrere Entwurfsentscheidungen zu fällen. Unter den Bedingungen des Fernzugriffs zu bibliographischen Datenbanken gibt es zwei Wege zur Installation zusätzlicher Intelligenz:

1. Manche Betreiber von Informationssystemen bieten seit längerer Zeit „Front-end-Prozessoren“ an, die auf der Client-Seite – also auf dem Personalcomputer des Benutzers – installiert werden und den Benutzer bei seiner Kommunikation mit dem IRS unterstützen.[1]

2. Andere Datenbank-Anbieter stellen traditionell auf der Server-Seite – also auf dem Computer, auf dem das IRS läuft - Zusatzsoftware bereit, durch die der Benutzer im Retrieval-Prozess geführt wird.[2]

Wir entscheiden uns für die Variante 2, wobei wir uns von folgenden Argumenten leiten lassen:

- Soll in die Kommunikation mit dem IRS Zusatzsoftware zur Realisierung von Methoden der künstlichen Intelligenz eingebunden werden, so sind aufwendige Algorithmen abzuarbeiten. Dafür sollte die große Prozessorleistung des Servers genutzt werden.

- Durch die Zusatzsoftware werden Algorithmen der Massendatenverarbeitung realisiert, die durch Lern- und Filterprozesse eine Informationsverdichtung herbeiführen. Um eine hohe Netzbelastung zu verhindern, ist es zweckmäßig, diese Prozesse am Ort der bibliographischen Datenbasen ablaufen zu lassen. Aufgabe des Client sollte es lediglich sein, die Ergeb-

[1] Vgl. beispielsweise Borgman/Case/Meadow (1989).
[2] Vgl. beispielhaft Vickery (1987).

nisse der Massendatenverarbeitung für den Benutzer in geeigneter Weise graphisch aufzubereiten.

- Die Entwicklung von Zusatzsoftware lohnt sich im allgemeinen nur dann, wenn sie für einen großen Benutzerkreis verfügbar gemacht wird. Die quasiparallele Verwendung von Programmen durch viele Benutzer wird nun aber auf dem Server in besonderer Weise unterstützt.

2.2 Programmschnittstelle oder programmierter Dialog?

Hat man sich dafür entschieden, die zusätzliche Intelligenz auf dem Server zu installieren, so sind komplizierte Probleme bei der Steuerungs- und Nachrichtenübergabe im Multi-user-Betrieb zu lösen. Für das Zusammenspiel von IRS und Zusatzsoftware stehen zwei Konzepte zur Verfügung:

1. Ein primitives Konzept besteht in der Nutzung von Programmschnittstellen: An definierten Punkten der Informationsverarbeitung kann das IRS veranlasst werden, die Steuerung an ein Anwenderprogramm abzutreten, das eine Verarbeitung der in der Schnittstelle übergebenen Daten vornimmt.

2. Ein produktiveres Konzept bieten jene IRS, die es Anwenderprogrammen gestatten, Kommandos an das IRS abzuschicken und die empfangenen Informationen weiterzuverarbeiten. Die Kommandos werden dabei in der Anfragesprache des IRS formuliert. In der normalerweise zwischen dem Benutzer und dem IRS stattfindenden Kommunikation wird dabei die Rolle des Fragenden vom Anwenderprogramm übernommen. Einen solchen Dialog bezeichnen wir als ***programmierten Dialog***.

Das Konzept des programmierten Dialogs ist Ausdruck eines qualitativ höheren Niveaus der Mensch-Maschine-Kommunikation, weil dem Benutzer nur noch solche Entscheidungen abverlangt werden, die nicht auch automatisch getroffen werden können.

Im Rahmen der Datenbanktechnologie wird dieses Prinzip bei den relationalen Datenbank-Managementsystemen (DBMS) mit ihrer Datenbanksprache SQL[1] verwirklicht. Ein prominenter Vertreter dieser Klasse von Datenbank-Manage-

[1] ***S***tructured ***Q***uery ***L***anguage - vgl. Date/Darwen (1998).

mentsystemen im PC-Sektor ist Access[1], das eine spezielle Programmiersprache (Visual Basic for Applications – VBA) zur Formulierung von Anwenderprogrammen bereitstellt. Noch bessere Möglichkeiten bietet beispielsweise ORACLE.[2] Dieses Datenbank-Managementsystem gestattet es, für die Formulierung von Anwenderprogrammen gängige Programmiersprachen - so etwa die prozeduralen Programmiersprachen C, COBOL oder FORTRAN sowie die objektorientierte Programmiersprache C++ - zu verwenden.

Die relationalen Datenbank-Managementsysteme erfordern und unterstützen das Konzept des programmierten Dialogs, weil die Daten strukturiert in Form von Tabellen vorliegen und meist in speziellen Anwenderprogrammen weiterverarbeitet werden sollen.[3] Bei der traditionellen Auswertung bibliographischer Datenbanken wird dagegen mit dem Auffinden relevanter Dokumente die Informationsversorgung des Benutzers als abgeschlossen betrachtet. Hinzu kommt, dass sich die bereitgestellten Texte im allgemeinen nicht so einfach auswerten lassen wie die Tabellenwerte aus relationalen Datenbanken.

Im vorliegenden Buch betrachten wir stellvertretend für bibliographische Informationssysteme solche Systeme, die mit einem prominenten Vertreter der Information-Retrieval-Systeme arbeiten: mit STAIRS[4] bzw. mit dessen Nachfolger SearchManager[5]. Auf Grund des hohen Bekanntheitsgrads des IRS STAIRS wollen wir als gemeinsame Bezeichnung für beide Systeme ***STAIRS*** verwenden.

STAIRS unterstützt eine Ankopplung von Anwenderprogrammen nur nach dem Konzept der Programmschnittstellen. Wollen wir jedoch unserer Zielstellung gerecht werden, zusätzliche Intelligenz in die Kommunikation mit dem Information-Retrieval-System STAIRS einzubringen, so müssen wir STAIRS dahingehend erweitern, dass auch das Konzept des programmierten Dialogs realisiert werden kann.

---

[1] Vgl. Feddema (2002).
[2] Vgl. Loney/Theriault (2001).
[3] Vgl. beispielsweise Date (1999) und Elmasri/Navathe (2002).
[4] ***ST***orage ***A***nd ***I***nformation ***R***etrieval ***S***ystem (IBM) - vgl. STAIRS (1981).
[5] Vgl. SearchManager (2006).

2.3 Unterprogramme oder Koprogramme?

Ehe die nächste Entscheidung hinsichtlich des Lösungswegs gefällt wird, müssen zunächst einige Fragen der Steuerungs- und Nachrichtenflüsse erläutert werden.

STAIRS ist ein Teilhabersystem, das mehreren Benutzern den quasiparallelen Zugriff zu bibliographischen Datenbanken ermöglicht. Das wird dadurch realisiert, dass STAIRS mit dem Transaktions-Manager CICS[1] zusammenarbeitet. Der Transaktions-Manager[2] enthält Mechanismen, die es ermöglichen, Anwenderprogramme im Rahmen eines Teilhabersystems zu betreiben. Unserem Anliegen, zusätzliche Intelligenz in Form eines KI-Assistenten in die Kommunikation mit einem IRS einzubinden, kommt es entgegen, dass CICS die Programmiersprache PL/I unterstützt.[3] Abbildung 1 veranschaulicht das Zusammenspiel zwischen den Anwenderprogrammen in PL/I[4], dem Transaktions-Manager CICS und dem Betriebssystem OS.

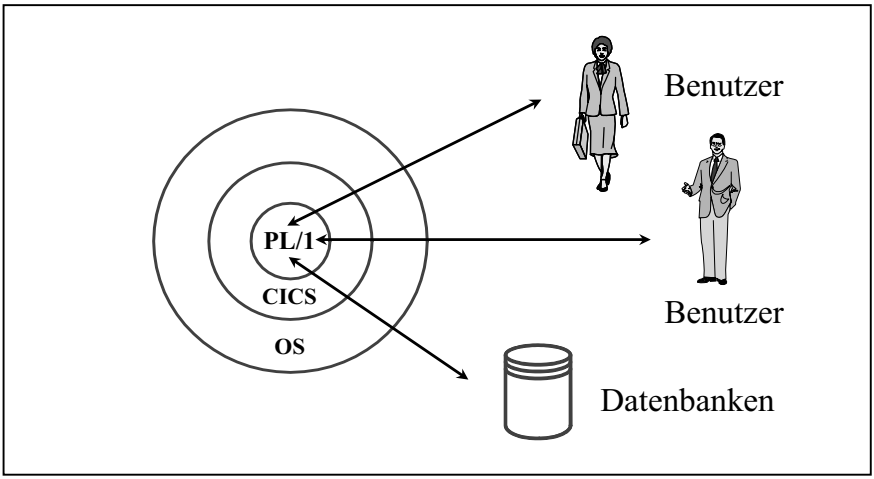


Abbildung 1: Teilhabersystem mit Anwenderprogrammen in PL/I unter Steuerung von CICS

[1] ***C***ustomer ***I***nformation ***C***ontrol ***S***ystem (IBM) - vgl. CICS (1982) bzw. CICS (2006).
[2] Zum Konzept des Transaktions-Managers vgl. Gray/Reuter (1993).
[3] CICS unterstützt außerdem die Programmiersprachen COBOL, C, C++ und Java.
[4] Obwohl PL/I schon 1964 entwickelt wurde, hat es auch heute noch Bedeutung, vgl. Sturm(2002).

Die Anwenderprogramme wenden sich mit ihren Anforderungen an Ressourcen und mit ihren Kommunikationswünschen ausschließlich an CICS, das die Synchronisation der Arbeit mehrerer Benutzer vornimmt, die Kommunikation abwickelt, Speicherbereiche verwaltet und Dateizugriffe realisiert. Dabei nimmt es die Dienste des Betriebssystems OS in Anspruch.

Beim Software-Entwurf von STAIRS legten die Entwickler besonderen Wert darauf, jede Abhängigkeit des IRS vom verwendeten Transaktions-Manager zu vermeiden. Sie erreichten das dadurch, dass alle Anforderungen an den Transaktions-Manager durch ein Steuerprogramm auf CICS abgebildet werden. Soll CICS durch einen anderen Transaktions-Manager abgelöst werden, so ist lediglich das Steuerprogramm zu verändern, während das IRS davon unberührt bleibt. Diese Architektur ist in Abbildung 2 dargestellt.

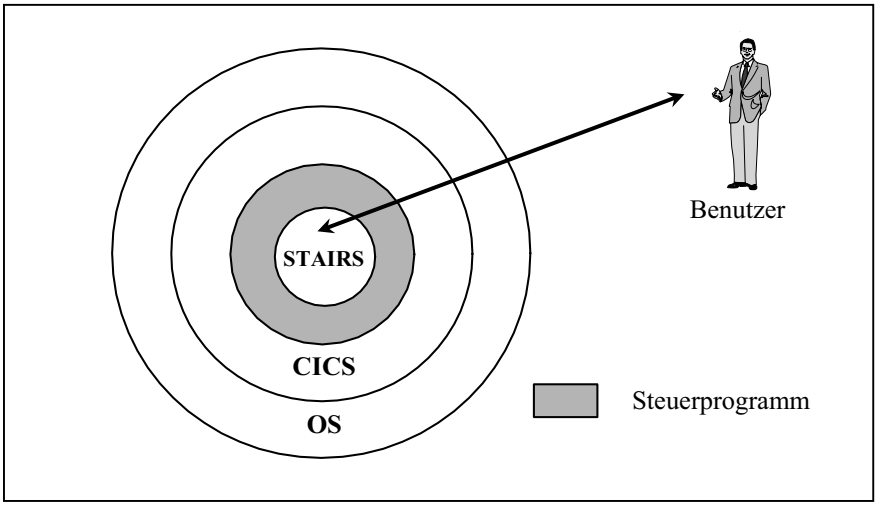


Abbildung 2: Zusammenspiel von STAIRS, Steuerprogramm und Transaktions-Manager CICS

Bei der Lösung der Aufgabe, zusätzliche Intelligenz in die Arbeit mit dem Information-Retrieval-System STAIRS einzubinden, wollen wir die Randbedingung einhalten, die erprobten Programmpakete STAIRS und CICS unverändert zu lassen. Dabei kommt uns die Schalenstruktur gemäß Abbildung 2 sehr entgegen,

weil wir so alle notwendigen Veränderungen auf das Steuerprogramm beschränken können.

Analysiert man die Erfahrungen bei der Entwicklung von Anwenderprogrammen, die unter CICS laufen sollen, so zeigt es sich, dass an den Programmierer hohe Anforderungen gestellt werden: Er muss nicht nur die Funktions- und Arbeitsweise von CICS kennen, sondern muss auch die Makroanweisungen von CICS im Detail beherrschen. Könnte man für die Anwenderprogramme die gleiche Unabhängigkeit von CICS erreichen, wie das im Falle von STAIRS gelungen ist, so würde man auch den Programmierer eines KI-Assistenten von diesen Detailkenntnissen entlasten.

Die geschilderten Überlegungen führen zu folgendem Lösungsansatz. Wir nutzen den aus Abbildung 2 ersichtlichen Umstand, dass die Kommunikation zwischen STAIRS und CICS durch das Steuerprogramm vermittelt wird: Dieses übernimmt vom einen Kommunikationspartner Steuerung und Nachricht und gibt beide an den anderen Kommunikationspartner weiter. Diese standardmäßig ablaufende Kommunikation wird nun modifiziert: Mit der Übergabe von Steuerung und Nachricht wird dem Steuerprogramm zugleich mitgeteilt, an wen es Steuerung und Nachricht weiterreichen soll. In das Steuerprogramm wird als dritter Kommunikationspartner eine zusätzliche Schale eingeschlossen, in der sich die Anwenderprogramme befinden. Diese können nun vom Steuerprogramm ebenfalls Steuerung und Nachricht erhalten. Sie führen eine Informationsverarbeitung durch und können ihrerseits das Steuerprogramm veranlassen, Steuerung und Nachricht entweder an STAIRS oder an CICS weiterzureichen. Das so in seinen Funktionen bereicherte Steuerprogramm nennen wir ***erweitertes Steuerprogramm***. Weil die im erweiterten Steuerprogramm eingeschlossene Schale von Anwenderprogrammen Algorithmen der Künstlichen Intelligenz realisieren soll, bezeichnen wir sie als ***KI-Assistenten***. Diese Systemarchitektur zeigt die Abbildung 3.

Bei der gewählten Systemarchitektur führt der KI-Assistent zwei Formen der Kommunikation durch:

1. Er kommuniziert mit dem Benutzer in einer semantisch orientierten Sprache, die auch der ungeübte Benutzer versteht. Wir bezeichnen das als ***fachspezifischen Dialog***.

2. Er kommuniziert mit dem Information-Retrieval-System STAIRS in der syntaktisch orientierten Anfragesprache, die von STAIRS vorgegeben ist, und realisiert so den ***programmierten Dialog***, von dem der Benutzer nun „verschont“ bleibt.

Vom erweiterten Steuerprogramm werden die Steuerungs- und Nachrichtenflüsse nicht nur eines einzigen Benutzers, sondern einer Vielzahl von Benutzern dirigiert. Dabei arbeitet jeder Benutzer unabhängig vom anderen. Jedem Benutzer muss somit auch die Möglichkeit eingeräumt werden, spezifische Anwenderprogramme in Anspruch zu nehmen. Zu jedem Zeitpunkt kann also ein Benutzer mit dem von ihm ausgewählten KI-Assistenten arbeiten.

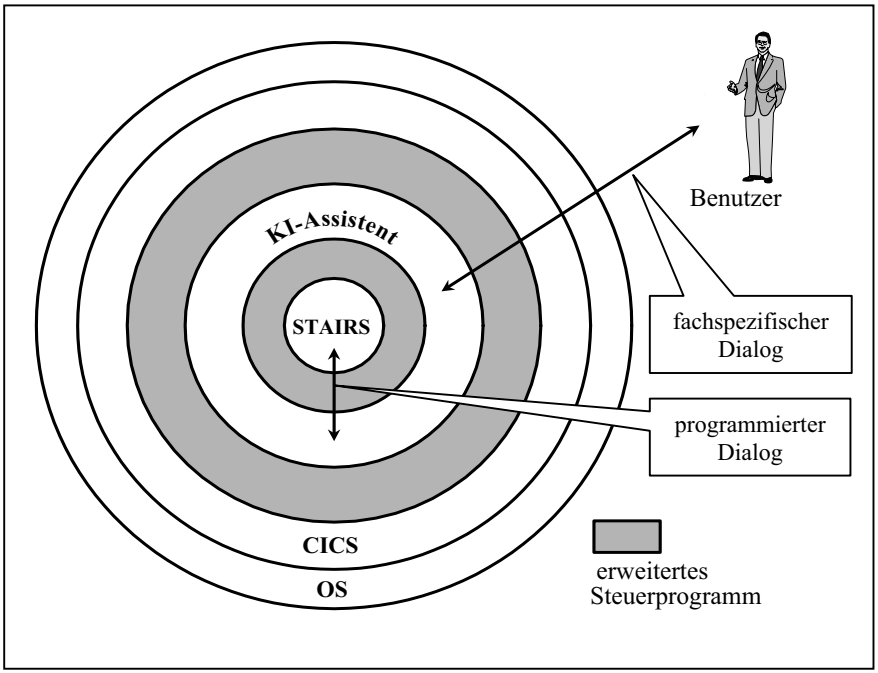


Abbildung 3: Zusammenspiel von STAIRS, erweitertem Steuerprogramm, KI-Assistent und CICS[1]

[1] Die Darstellung erfolgt aus der Sicht eines einzelnen Benutzers; sie zeigt deshalb nur einen einzigen KI-Assistenten.

Wir stehen nun vor der Entscheidung, in welcher Weise die Steuerungsübergabe zwischen den Anwenderprogrammen des KI-Assistenten realisiert werden soll. Würde man sich nur auf die in CICS vorgefertigten Mechanismen der Steuerungsübergabe beschränken, so wären lediglich einfache Unterprogramm-Aufrufe möglich. Wir benötigen für die Programmierung zusätzlicher Intelligenz jedoch leistungsfähigere Mechanismen. Wir entscheiden uns deshalb dafür, kompliziertere Steuerungsflüsse zu ermöglichen, um Elemente der objektorientierten Programmierung - abstrakte Datentypen und Koprogramme - realisieren zu können.

Um die komplizierteren Steuerungsflüsse im Rahmen eines programmierten Dialogs verwirklichen zu können, sind drei Richtungen des Lösungsansatzes weiterzuverfolgen:

- Im Kapitel 3 wird ein theoretisches Modell entwickelt, das die Steuerungs- und Nachrichtenflüsse der Anwenderprogramme des KI-Assistenten untereinander und mit dem IRS beschreibt.

- Im Kapitel 4 erfolgt der Entwurf einer Programmiersprache zur Realisierung des theoretischen Modells.

- Das Kapitel 5 demonstriert die Möglichkeiten zum Entwurf zusätzlicher Intelligenz auf der Grundlage von Lernalgorithmen und Algorithmen der Massendatenverarbeitung.

3 Modell der Einbindung eines KI-Assistenten

3.1 Überblick über die Steuerungsflüsse

Um die komplizierten Steuerungsflüsse im Rahmen eines programmierten Dialogs beschreiben zu können, muss ein theoretisches Modell entwickelt werden, das eine Synthese aus folgenden drei Entwicklungsrichtungen herbeiführt:

1. problemspezifischer Dialog der Anwenderprogramme des KI-Assistenten mit dem Benutzer,

2. programmierter Dialog zwischen den Anwenderprogrammen und dem IRS,

3. ereignisabhängige Steuerungs- und Nachrichtenübergabe zwischen den Anwenderprogrammen.

Durch diese Synthese soll gegenüber dem ursprünglich verfügbaren Stand der Software-Entwicklung für das Information-Retrieval-System STAIRS ein neues Niveau erreicht werden.

Wir betrachten zunächst das originäre Zusammenspiel des Benutzers mit dem Information-Retrieval-System STAIRS und beschreiben dann die von uns vorgenommenen Modifizierungen.

Das Information-Retrieval-System STAIRS ist ein Teilhabersystem. Die quasiparallele Arbeit mehrerer Benutzer wird durch den Transaktions-Manager CICS realisiert. Ein Programm von STAIRS arbeitet zu jedem Zeitpunkt stets nur für einen Benutzer. Die Möglichkeit seiner Mehrfachanwendung für viele Benutzer ergibt sich aus der Tatsache, dass es reenterabel programmiert wurde. Wir können uns unter der Voraussetzung, dass alle Anwenderprogramme für den programmierten Dialog ebenfalls reenterabel gehalten werden, darauf beschränken, das Zusammenspiel nur eines Benutzers mit STAIRS zu beschreiben, und müssen dann CICS nicht mehr explizit erwähnen.

Wenn STAIRS dem Benutzer Informationen bereitstellt und ihn zur Eingabe eines Kommandos auffordert, finden die Steuerungsübergaben statt, die in der Abbildung 4 dargestellt sind:

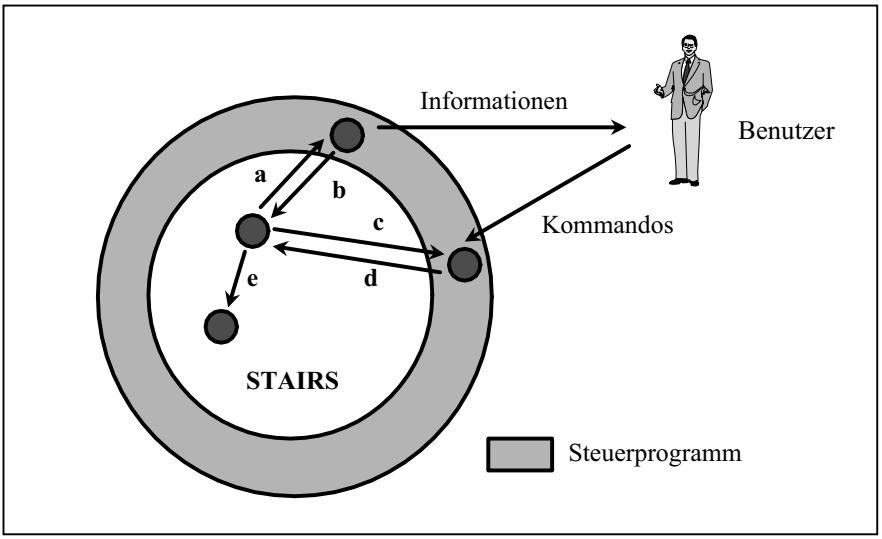


Abbildung 4: Originäres Zusammenspiel von STAIRS mit dem Steuerprogramm

a: STAIRS unterbricht seine Arbeit und gibt die Informationen und die Steuerung an das Steuerprogramm weiter. STAIRS verharrt in seinem Bearbeitungsstand, wobei alle benutzerspezifischen Daten erhalten bleiben.

b: Das Steuerprogramm sorgt dafür, dass die von STAIRS bereitgestellten Informationen zum PC des Benutzers geleitet werden. Dann gibt es die Steuerung an STAIRS zurück.

c: STAIRS wendet sich mit der Aufforderung an das Steuerprogramm, das nächste Kommando des Benutzers bereitzustellen.

d: Das Steuerprogramm beobachtet den PC des Benutzers. Hat dieser sein Kommando abgeschickt, so wird es vom Steuerprogramm entgegengenom-

men. Dann übergibt das Steuerprogramm das Kommando des Benutzers und die Steuerung wieder an STAIRS.

e: STAIRS setzt die Informationsverarbeitung für den Benutzer an der Stelle fort, an der sie im Punkt a unterbrochen wurde.

Sowohl beim Absenden von Informationen an den Benutzer als auch beim Anfordern eines Kommandos vom Benutzer gibt STAIRS die Steuerung an das Steuerprogramm ab. Das Steuerprogramm führt mit den Informationen und Kommandos nur solche Operationen aus, die mit der Darstellung und Übertragung der Nachrichten zusammenhängen (Formatierung, Einfügen von Steuerzeichen usw.).

An diesem Punkt setzt unsere Lösung zur Einbindung zusätzlicher Intelligenz in die Kommunikation mit dem IRS ein. Wir sichern durch eine Modifizierung des Steuerungsflusses, dass die Nachrichten problemspezifischen Transformationen und Verarbeitungen unterzogen werden können. Dazu wird das Steuerprogramm dahingehend erweitert, dass die Steuerungsübergaben ablaufen können, die in der Abbildung 5 veranschaulicht werden:

a: STAIRS unterbricht seine Arbeit. Die Informationen und die Steuerung werden an das - nunmehr erweiterte - Steuerprogramm übergeben.

b: Das erweiterte Steuerprogramm nimmt die Informationen zwar entgegen, leitet sie aber nicht an den Benutzer weiter. Es gibt die Steuerung an STAIRS zurück.

c: STAIRS wendet sich mit der Aufforderung an das erweiterte Steuerprogramm, das nächste Kommando des Benutzers bereitzustellen.

d: Das erweiterte Steuerprogramm übergibt die im Punkt a empfangenen Informationen von STAIRS gemeinsam mit der Steuerung an ein Anwenderprogramm des KI-Assistenten.

e: Das Anwenderprogramm wertet die Informationen problemspezifisch aus. Dabei kann es mit weiteren Anwenderprogrammen zusammenarbeiten. Die Gesamtheit der Anwenderprogramme bildet den KI-Assistenten. Dieser kann im Zuge seiner Arbeit Kommunikation mit dem Benutzer führen. Dies erfolgt durch Vermittlung des erweiterten Steuerprogramms. Er kann natür-

lich auch problemspezifische Daten speichern. Schließlich gibt ein Programm des KI-Assistenten gemeinsam mit einem Kommando für STAIRS die Steuerung an das erweiterte Steuerprogramm zurück. Dann verharren alle Programme des KI-Assistenten in ihrem Verarbeitungszustand. Dabei bleiben die problemspezifischen Daten erhalten.

f: Das erweiterte Steuerprogramm übergibt das Kommando und die Steuerung an STAIRS.

g: STAIRS setzt die Informationsverarbeitung an der Stelle fort, an der sie im Punkt a unterbrochen wurde. Für STAIRS bleibt es unerkannt, dass es nun nicht ein Kommando des Benutzers, sondern ein Kommando des KI-Assistenten ausführt.

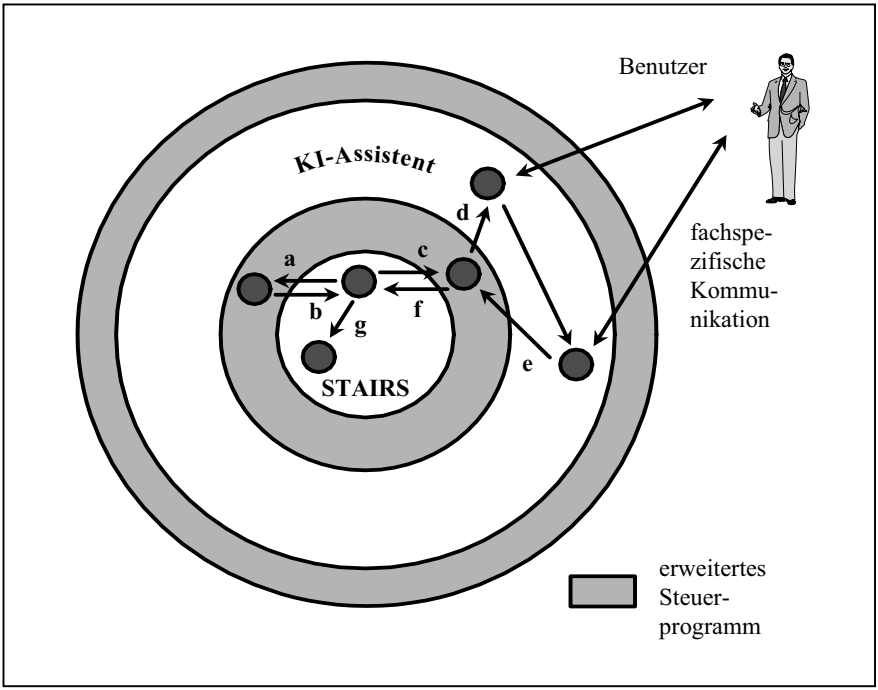


Abbildung 5: Zusammenspiel von STAIRS mit dem KI-Assistenten

Durch diesen Lösungsansatz wird gesichert, dass die Einbindung zusätzlicher Intelligenz in die Kommunikation mit dem IRS möglich wird, ohne dass in die Programme von STAIRS eingegriffen werden muss. Lediglich das Steuerprogramm muss erweitert werden, um den modifizierten Steuerungsfluss realisieren zu können.

Zur Verwirklichung des beschriebenen Lösungsansatzes sind nun zwei wichtige Probleme zu bearbeiten:

1. Entwicklung eines Modells für die Steuerungs- und Kommunikationsflüsse,

2. Entwurf und Implementierung einer Programmiersprache zur Formulierung von Anwenderprogrammen.

Das erste Problem wird im Rest dieses Kapitels behandelt, während die Programmiersprache im Kapitel 4 vorgestellt wird.

3.2 Modell der Steuerungsflüsse

Zur Verwirklichung des beschriebenen Lösungsansatzes soll nun ein Modell entwickelt werden, nach dem die Steuerungsübergabe zwischen den Programmen des KI-Assistenten untereinander und mit dem Information-Retrieval-System STAIRS erfolgt. Dabei werden vier Entwurfsziele verfolgt:

1. modularer Aufbau des KI-Assistenten,

2. Verbinden des Steuerungsflusses mit einer Informationsübergabe,

3. Gestaltung von Algorithmen, die als Routinen oder Prozesse arbeiten,

4. Realisierung von abstrakten Datentypen und Koprogrammen.

Wir bilden die Steuerungsflüsse zwischen den Programmen des KI-Assistenten untereinander und mit STAIRS in einem zeitabhängigen System ab: Das System $\mathcal{M}^t$ besteht zum Zeitpunkt t aus einer Menge ***abstrakter Maschinen*** M^t, über der zwei Relationen γ^t und α^t definiert sind:

$$\begin{aligned} \mathcal{M}^t &= \left(M^t, \gamma^t, \alpha^t \right) \\ \gamma^t &\subset M^t \times M^t \\ \alpha^t &\subset M^t \times M^t \end{aligned} \tag{1}$$

Eine abstrakte Maschine $m \in M^t$ ist gegeben durch einen Algorithmus $\mathcal{A}$, nach dem sie arbeitet, und durch ein Datenobjekt d, das durch den Algorithmus $\mathcal{A}$ verändert wird: $m = (\mathcal{A}, d)$.

Wir abstrahieren in unserem Modell von der konkreten Ausprägung des Algorithmus $\mathcal{A}$. Als wesentlich betrachten wir nur, dass er eine Menge von Punkten enthält, an denen eine Nachricht an einen Empfänger abgeschickt und diesem gleichzeitig die Steuerung übergeben wird. Einen solchen Punkt bezeichnen wir als ***Sendepunkt*** $s_{\mathcal{A}}$:

$$\mathcal{A} = \left\{ s_{\mathcal{A}1}, s_{\mathcal{A}2}, \ldots, s^+_{\mathcal{A}} \right\} \tag{2}$$

Jeder Algorithmus hat wenigstens einen Sendepunkt $s^+_{\mathcal{A}}$, den wir als seinen ***Endpunkt*** bezeichnen. In diesem Punkt endet der Informationsverarbeitungsprozess, der durch den Algorithmus $\mathcal{A}$ beschrieben wird. Die Steuerung muss dann einer anderen Maschine[1] übertragen werden. Das erfolgt gemeinsam mit dem Senden einer - eventuell leeren - Nachricht.

In jedem Sendepunkt $s_{\mathcal{A}}$ eines Algorithmus $\mathcal{A}$ wird eine Nachricht n an einen Empfänger abgeschickt, der durch eine Empfängerangabe c gekennzeichnet ist:

[1] Wenn Missverständnisse ausgeschlossen sind, werden wir vereinfachend statt von ***abstrakten Maschinen*** einfach von ***Maschinen*** sprechen.

$$s_{\mathcal{A}} = (n, c) \in \mathcal{A} \tag{3}$$

Die Nachricht n informiert den Empfänger c über den Stand der Abarbeitung des Algorithmus $\mathcal{A}$, der im Sendepunkt $s_{\mathcal{A}}$ erreicht wurde.

Die Maschinen des Systems $\mathcal{M}^t$ treten in den Sendepunkten ihrer Algorithmen miteinander in Beziehung und ordnen sich dabei in die Relationen γ^t bzw. α^t ein.

Ein Element $(m_g, m) \in \gamma^t$ bringt zum Ausdruck, dass die Maschine m_g die Maschine m ins Leben gerufen hat. Wir bezeichnen deshalb m_g als den ***Generator*** von m. Bei der Beschreibung des Zeitverhaltens des Systems $\mathcal{M}^t$ spielt diese Relation dann eine Rolle, wenn die Maschine m aufhört zu existieren, wenn sie also ihren Endpunkt erreicht hat.

Ein Element $(m_a, m) \in \alpha^t$ bringt zum Ausdruck, dass die Maschine m_a die Maschine m mit dem Zusenden einer Nachricht zur Fortsetzung ihrer Arbeit aufgefordert, sie also wieder aktiviert hat. Wir bezeichnen deshalb m_a als den ***Aktivator*** von m. Bei der Beschreibung des Zeitverhaltens des Systems $\mathcal{M}^t$ spielt diese Relation eine Rolle, wenn m in einem Sendepunkt eine Nachricht an den Aktivator zurückschicken möchte.

Zum Zeitpunkt t_0 veranlasst das erweiterte Steuerprogramm den Aufbau des Systems der abstrakten Maschinen in seinem Initialzustand $\mathcal{M}^{t_0}$. Die Menge M^{t_0} enthält dann zwei Maschinen:

1. Die Maschine 'IRS', die nach dem Algorithmus des Information-Retrieval-Systems STAIRS arbeitet. Dabei wird von der modularen Struktur des Pro-

grammsystems STAIRS abstrahiert und dieses als undifferenzierter Algorithmus betrachtet.

2. Die Maschine m_0, deren Algorithmus der Benutzer auswählt. Diese Maschine übernimmt die Regie über die Arbeit des KI-Assistenten. Wir bezeichnen sie deshalb als ***Monitormaschine***.

Zwischen den Maschinen 'IRS' und m_0 besteht zunächst noch keine Beziehung, so dass die Relationen γ^{t_0} und α^{t_0} leer sind:

$$\mathcal{M}^{t_0} = \left(\{\text{'IRS'}, m_0\}, \varnothing, \varnothing\right) \tag{4}$$

Das weitere Zeitverhalten des Systems der abstrakten Maschinen wird durch die Aktivitäten bestimmt, die in den Sendepunkten der Algorithmen ausgeführt werden. Bei der Darlegung dieser Aktivitäten werden wir folgende Arten von Sendepunkten unterscheiden:

1. Sendepunkte, in denen eine neue Maschine eingerichtet und aktiviert wird,

2. Sendepunkte, in denen eine bereits existierende Maschine zur Fortsetzung ihrer Arbeit aufgefordert wird,

3. Sendepunkte, in denen eine Maschine aufhört zu existieren.

Das Erweitern des Systems $\mathcal{M}^{t-1}$ zum Zeitpunkt t erfolgt, indem eine neue Maschine zum KI-Assistenten hinzugefügt wird.[1] Diesen Vorgang bezeichnen wir als ***Generierung***. Soll von einer Maschine m_g, die nach dem Algorithmus $\mathcal{A}_g$ arbeitet, eine neue Maschine generiert werden, die nach einem Algorith-

[1] Durch die Schreibweise *t-1* wird der Zeitpunkt ausgedrückt, an dem das System der abstrakten Maschinen seinen gegenwärtigen Zustand angenommen hat. Dieser wird nun zum Zeitpunkt *t* verändert.

mus $\mathcal{A}$ arbeiten soll, so muss $\mathcal{A}_g$ einen Sendepunkt enthalten, den wir als ***Generierungspunkt***

$$s_g = \left(n\,,\, \underline{\text{gen}}\left(\mathcal{A}, I(m)\right)\right) \in \mathcal{A}_g \tag{5}$$

bezeichnen. Das Absenden der Nachricht n durch die Maschine m_g in einem Generierungspunkt löst die folgenden Aktivitäten des erweiterten Steuerprogramms aus:

1. Eine neue Maschine m wird eingerichtet, die ein Datenobjekt d nach dem Algorithmus $\mathcal{A}$ bearbeitet. Die neue Maschine wird zum KI-Assistenten hinzugefügt. Das bedeutet eine Erweiterung der Menge der Maschinen, die zum Zeitpunkt $t-1$ existieren:

$$M^t = M^{t-1} \cup \{m\};\ m = \left(\mathcal{A}, d\right) \tag{6}$$

   Zur Identifizierung der neuen Maschine m wird ihr vom erweiterten Steuerprogramm ein Identifikator $I(m)$ zugeordnet, der dem Generator m_g mitgeteilt wird. Welche erweiterten Möglichkeiten daraus erwachsen, wenn die Maschine m_g diesen Identifikator auch anderen Maschinen des KI-Assistenten mitteilt, werden wir in den folgenden Abschnitten sehen.

2. Die Maschine m_g ist der Generator der neu eingerichteten Maschine m. Deshalb wird die Relation γ^{t-1} um ein neues Element erweitert:

$$\gamma^t = \gamma^{t-1} \cup \{(m_g, m)\} \tag{7}$$

3. Das erweiterte Steuerprogramm übergibt der neu eingerichteten Maschine m gemeinsam mit der Nachricht n auch die Steuerung. Dadurch wird sie aktiviert. Deshalb wird auch die Relation α^{t-1} um ein neues Element erweitert:

$$\alpha^t = \alpha^{t-1} \cup \{(m_g, m)\} \tag{8}$$

Die Maschine m arbeitet nun solange nach dem Algorithmus $\mathcal{A}$, bis dieser seinen ersten Sendepunkt erreicht.

Der Generator m_g verharrt im Generierungspunkt, bis er die Steuerung wiedererhält.

Am Beginn dieses Abschnitts wurde das Entwurfsziel formuliert, in die Realisierung eines KI-Assistenten abstrakte Datentypen und Koprogramme einzubeziehen. Wir beschränken uns deshalb nicht auf das einfache Konzept der Unterprogramm-Technik, bei dem die aufgerufene Komponente die Steuerung stets wieder an die aufrufende Komponente zurückgeben muss. In unserem Modell besteht zwischen m_g und m kein strenges Unterordnungsverhältnis. Die Maschine m kann die Steuerung an dritte Maschinen des Systems weitergeben, so dass m_g letztlich von jeder beliebigen Maschine des Systems die Steuerung zurückerhalten kann. Die sich daraus ergebenden Konsequenzen werden wir im Abschnitt 3.3 näher erläutern.

Zu jedem beliebigen Zeitpunkt t ist nur eine Maschine m_a des Systems $\mathcal{M}^t$ aktiv, während alle anderen passiv sind und in ihrem Bearbeitungsstand verharren. Die aktive Maschine m_a, die nach dem Algorithmus $\mathcal{A}_a$ arbeitet, kann in einem Sendepunkt ihre Arbeit unterbrechen und gemeinsam mit einer Nach-

richt n die Steuerung einer beliebigen anderen passiven Maschine m übergeben. Diesen Vorgang nennen wir ***Aktivierung***. Der Sendepunkt, in dem das geschieht, wird als ***Aktivierungspunkt*** bezeichnet:

$$s_a = \left(n, \underline{act}(I(m))\right) \in \mathcal{A}_a \tag{9}$$

Zur Aktivierung von m muss die Maschine m_a dem erweiterten Steuerprogramm den Identifikator von m bekannt geben. Ist m_a der Generator von m, so wurde ihm $I(m)$ im Zuge des Generierens mitgeteilt. In allen anderen Fällen muss $I(m)$ der Maschine m_a zuvor durch Nachrichtenübermittlung in den Sendepunkten übergeben worden sein.

Sendet die Maschine m_a in einem Aktivierungspunkt eine Nachricht n an die Maschine m ab, so löst sie die folgenden Aktivitäten des erweiterten Steuerprogramms aus:

1. Die Relation α^{t-1} wird aktualisiert. Enthält sie ein Element $(x,m) \in \alpha^{t-1}$, so wird dieses aus der Relation entfernt. Das Element (m_a, m) wird zur Relation hinzugefügt:

$$\alpha^t = \alpha^{t-1} - \left\{ (x,m) \,\middle|\, (x,m) \in \alpha^{t-1} \right\} \cup \{ (m_a, m) \} \tag{10}$$

   Durch diese Vorgehensweise wird gesichert, dass die Relation α^t zu jedem Zeitpunkt t nur einen einzigen Aktivator von m ausweist.

2. Gemeinsam mit der Nachricht n wird der Maschine m die Steuerung übergeben. Die Maschine m setzt ihre Arbeit an der Stelle fort, an der sie diese zuletzt unterbrochen hat. Sie bleibt solange aktiv, bis sie ihren nächsten Sendepunkt erreicht hat.

Ein besonderer Fall liegt dann vor, wenn die aktive Maschine m eine passive Maschine m_a aktivieren will und m_a diejenige Maschine ist, die m das letzte Mal aktiviert hat. Das entspricht der Situation, dass m von m_a einen Auftrag zur Bearbeitung erhalten hat und nun die Bearbeitungsergebnisse zurückmelden möchte. Häufig sind jedoch die Aufträge so beschaffen, dass zu ihrer Ausführung die Kenntnis des Auftraggebers nicht erforderlich ist. Um auch dann noch das Zurückmelden der Ergebnisse zu gewährleisten, wurde die Relation α^t in das System $\mathcal{M}^t$ aufgenommen. Einen Sendepunkt, an dem die nach dem Algorithmus $\mathcal{A}$ arbeitende Maschine m die erzielten Ergebnisse in Form der Nachricht n an ihren Aktivator zurücksenden möchte, bezeichnen wir als ***Aktivator-Rückkehrpunkt***:

$$s_{ar} = \left(n, \underline{aret}\right) \in \mathcal{A} \tag{11}$$

Dabei werden vom erweiterten Steuerprogramm folgende Aktivitäten ausgeführt:

1. Der Aktivator m_a wird auf Grund des Elements $(m_a, m) \in \alpha^{t-1}$ ermittelt und dieses Element aus der Relation α^{t-1} entfernt:

$$\alpha^t = \alpha^{t-1} - \{(m_a, m)\} \tag{12}$$

2. Gemeinsam mit der Nachricht n wird die Steuerung an die Maschine m_a übergeben.

Wir haben damit den Steuerungsmechanismus entwickelt, nach dem Maschinen eingerichtet werden und sie sich gegenseitig aktivieren. Nun bleibt nur noch zu beschreiben, wie Maschinen aus dem System $\mathcal{M}^t$ entfernt werden.

Erreicht eine Maschine m, die nach dem Algorithmus $\mathcal{A}$ arbeitet, schließlich ihren Endpunkt $s^+_{\mathcal{A}}$, so hat sie die ihr übertragene Aufgabe gelöst und kann aus dem KI-Assistenten entfernt werden. Da m im Besitz der Steuerung ist, muss sie diese gemeinsam mit einer Nachricht n einer anderen Maschine m' übergeben. Dazu kann m die Maschine m' explizit durch Angabe des Identifikators $I(m')$ auswählen. Der Endpunkt hat dann die Form:

$$s^+_{\mathcal{A}} = \left(n, \underline{act^+}(I(m')) \right) \in \mathcal{A} \tag{13}$$

Die Maschine m kann die Steuerung aber auch an eine Maschine übergeben, zu der sie in einer besonderen Relation steht - nämlich an ihren Generator oder an ihren Aktivator. Wir schreiben das in der Form

$$s^+_{\mathcal{A}} = \left(n, \underline{gret^+} \right) \in \mathcal{A} \tag{14}$$

beziehungsweise in der Form

$$s^+_{\mathcal{A}} = \left(n, \underline{aret^+} \right) \in \mathcal{A} \tag{15}$$

Erreicht die Maschine m ihren Endpunkt, so führt das erweiterte Steuerprogramm folgende Aktivitäten aus:

1. Die Maschine m wird aus dem System $\mathcal{M}^{t-1}$ entfernt:

$$M^t = M^{t-1} - \{m\} \tag{16}$$

2. Soll die Steuerung an den Generator zurückgegeben werden, so wird der Generator $m' = m_g$ auf Grund des Elements $(m_g, m) \in \gamma^{t-1}$ ermittelt. Soll dagegen dem Aktivator $m' = m_a$ die Steuerung übergeben werden, so wird dieser auf Grund des Elementes $(m_a, m) \in \alpha^{t-1}$ bestimmt. Ansonsten hat m die Maschine, der die Steuerung zu übergeben ist, mit $I(m')$ explizit festgelegt.

3. Aus den Relationen γ^{t-1} und α^{t-1} werden alle Hinweise auf die Beziehungen gelöscht, die die Maschine m zu anderen Maschinen einging:

$$\begin{aligned} \gamma^t &= \gamma^{t-1} - \left\{(m,x) \middle| (m,x) \in \gamma^{t-1}\right\} - \left\{(m_g, m)\right\} \\ \alpha^t &= \alpha^{t-1} - \left\{(m,x) \middle| (m,x) \in \alpha^{t-1}\right\} - \left\{(m_a, m)\right\} \end{aligned} \tag{17}$$

4. Die Steuerung wird gemeinsam mit der Nachricht n der Maschine m' übergeben.

Das Zeitverhalten des Systems $\mathcal{M}^t$ lässt sich zusammenfassend in Abhängigkeit von den Ereignissen E1 bis E5 wie folgt darstellen:

$$\mathcal{M}^t = \begin{cases} (\{\text{'IRS'}, m_0\}, \varnothing, \varnothing) & \text{E1} \\ \left(M^{t-1} \cup \{m\}, \gamma^{t-1} \cup \left\{(m_g, m)\right\}, \alpha^{t-1} \cup \left\{(m_g, m)\right\}\right) & \text{E2} \\ \left(M^{t-1}, \gamma^{t-1}, \alpha^{t-1} - \left\{(x,m) \,\middle|\, (x,m) \in \alpha^{t-1}\right\} \cup \{(m_a, m)\}\right) & \text{E3} \\ \left(M^{t-1}, \gamma^{t-1}, \alpha^{t-1} - \{(m_a, m)\}\right) & \text{E4} \\ \begin{pmatrix} M^{t-1} - \{m\}, \gamma^{t-1} - \left\{(m,x) \mid (m,x) \in \gamma^{t-1}\right\} - \left\{(m_g, m)\right\}, \\ \alpha^{t-1} - \left\{(m,x) \mid (m,x) \in \alpha^{t-1}\right\} - \{(m_a, m)\} \end{pmatrix} & \text{E5} \end{cases}$$

Bei den Ereignissen E1 bis E5 handelt es sich um die folgenden Situationen:

E1: Der KI-Assistent wird eingerichtet.

E2: Die Maschine m_g generiert eine neue Maschine m. Dabei wird m zugleich aktiv.

E3: Die Maschine m_a aktiviert die existierende Maschine m.

E4: Die Maschine m gibt die Steuerung an ihren Aktivator m_a zurück.

E5: Die Maschine m hört auf zu existieren. Ihre Beziehungen zum Generator, zum Aktivator und zu anderen Maschinen werden in den Relationen gelöscht.

Das Verhalten des Systems $\mathcal{M}^t$ lässt sich in anschaulicher Weise mit Hilfe eines Sequenzdiagramms der objektorientierten Modellierungssprache UML[1] darstellen. Die Abbildung 6 zeigt einen beispielhaften Zeitverlauf.

[1] ***U**nified **M**odeling **L**anguage* - vgl. beispielsweise Störrle (2005).

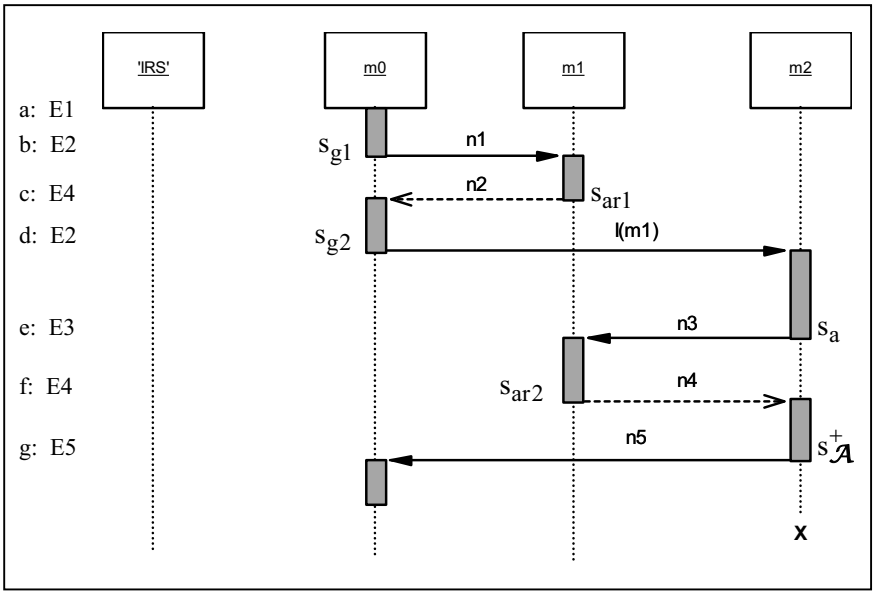


Abbildung 6: Zeitverhalten des Systems $\mathcal{M}^t$ im Sequenzdiagramm der UML

Die senkrechten Linien des Sequenzdiagramms symbolisieren die Lebenslinien des Information-Retrieval-Systems STAIRS ('IRS'), der Monitormaschine (m0) und der weiteren Maschinen m1 und m2. Jede Lebenslinie stellt einen (logischen) Zeitstrahl von oben nach unten dar.

Im Zeitverlauf finden die folgenden Ereignisse statt:

a: Ereignis E1: Der KI-Assistent wird eingerichtet. Er enthält neben dem vorerst passiven Information-Retrieval-System ('IRS' - gepunktete Lebenslinie) die aktive Monitormaschine m0, der vom erweiterten Steuerprogramm die Steuerung übergeben wurde (Aktivierungsbalken auf der Lebenslinie).

b: Ereignis E2: Die Monitormaschine m0 ruft im Generierungspunkt s_{g1} die Maschine m1 ins Leben, übermittelt ihr die Nachricht n1 und aktiviert sie (durchgezogener Aktivierungspfeil von m0 nach m1). Das

erweiterte Steuerprogramm übergibt der generierenden Maschine $m0$ den Identifikator $I(m1)$ der neu eingerichteten Maschine.

c: Ereignis E4: Die Maschine $m1$ erreicht ihren ersten Aktivator-Rückkehrpunkt s_{ar1}. Sie gibt gemeinsam mit der Nachricht $n2$ die Steuerung an ihren Aktivator zurück, also an die Monitormaschine $m0$ (gestrichelter Return-Pfeil). Diese ist nun wieder aktiv (Wiederaufnahme des Aktivierungsbalkens auf ihrer Lebenslinie).

d: Ereignis E2: Die Monitormaschine $m0$ ruft in einem zweiten Generierungspunkt s_{g2} die Maschine $m2$ ins Leben, übermittelt ihr als Nachricht den Identifikator $I(m1)$ der Maschine $m1$ und aktiviert sie. Das erweiterte Steuerprogramm teilt $m0$ den Identifikator $I(m2)$ der neu eingerichteten Maschine $m2$ mit.

e: Ereignis E3: Die Maschine $m2$, die ja bei Ihrer Generierung den Identifikator $I(m1)$ der Maschine $m1$ erhalten hat, übergibt im Aktivierungspunkt s_a der passiven Maschine $m1$ gemeinsam mit der Nachricht $n3$ die Steuerung.

f: Ereignis E4: Im Aktivator-Rückkehrpunkt s_{ar2} wird die Maschine $m1$ passiv und aktiviert $m2$ unter Zusendung der Nachricht $n4$.

g: Ereignis E5: Die Maschine $m2$ erreicht ihren Endpunkt $s^+_{\mathcal{A}}$. Sie hört auf zu existieren (Kreuz am Ende der Lebenslinie) und übergibt ihrem Generator $m0$ gemeinsam mit der Nachricht $n5$ die Steuerung.

Das System der abstrakten Maschinen wurde als ein zeitabhängiges System beschrieben. Das hat den folgenden Grund. Der KI-Assistent wird durch Algorithmen realisiert, die in einem Einprozessor-Computer sequentiell abgearbeitet werden. Daraus ergibt sich die Zweckmäßigkeit, den Gesamtalgorithmus in Moduln zu zerlegen, die nicht alle gleichzeitig im Hauptspeicher des Computers verfügbar sein müssen. In unserem Modell werden diese Moduln den abstrakten Maschinen zugeordnet. Die Unabhängigkeit der einzelnen Moduln voneinander

wird dadurch gesichert, dass die Maschinen lediglich Nachrichten untereinander austauschen.

Mit der modularen Architektur des KI-Assistenten wird zugleich auch die Aufgabe gelöst, die problemspezifischen Daten zu portionieren. Eine Maschine $m = (\mathcal{A}, d)$ wird erst zu dem Zeitpunkt generiert, von dem an sie gebraucht wird. Erst dann wird auch der Speicherbereich für das Datenobjekt d bereitgestellt. Bei der Realisierung des Modells muss gesichert werden, dass das Datenobjekt d in seiner Größe zeitlich variieren kann. Eine Maschine muss also in der Lage sein, ihr Datenobjekt bei Bedarf zu vergrößern und entsprechend wieder zu verkleinern. Erst mit dem Entfernen der Maschine m aus dem KI-Assistenten wird das Datenobjekt d aufgegeben.

Anders liegen die Verhältnisse beim Algorithmus $\mathcal{A}$. Während jede Maschine ihr individuelles Datenobjekt besitzt, wird der Algorithmus von mehreren Maschinen gemeinsam genutzt. Das erweiterte Steuerprogramm sorgt dafür, dass ein Algorithmus $\mathcal{A}$ nur bei Bedarf und nur in einem einzigen Exemplar in den Hauptspeicher geladen wird. Die Möglichkeit der Mehrfachnutzung des Algorithmus wird - wie schon erwähnt - dadurch gesichert, dass ein Programm reenterabel ist. Jede Maschine, die nach dem Algorithmus $\mathcal{A}$ arbeitet, muss sich jedoch in ihrem individuellen Datenobjekt „merken“, wie weit sie in der Abarbeitung des Algorithmus vorangeschritten ist. Das erweiterte Steuerprogramm übernimmt diese organisatorische Aufgabe, so dass sie bei der Programmierung der Algorithmen nicht berücksichtigt werden muss.

3.3 Routinen und Prozesse

Im vorangegangenen Abschnitt wurde das Modell entwickelt, nach dem die Steuerungs- und Nachrichtenübergabe zwischen abstrakten Maschinen erfolgen soll. In Vorbereitung auf die Realisierung dieses Modells durch eine Programmiersprache wollen wir nun typische Steuerungsabläufe beschreiben und dabei die Algorithmen in Routinen und Prozesse einteilen.

Eine Maschine $m = (\mathcal{A}, d)$ kann die Ausführung einer abgegrenzten Aufgabe einer anderen Maschine $m' = (\mathcal{A}', d')$ übertragen. Die Aufgabe sei so beschaffen, dass die Maschine m' keine Zwischenergebnisse, sondern lediglich das

Endergebnis zu liefern hat. Dann will die Maschine m die Steuerung gemeinsam mit dem Endergebnis erst dann wieder zurückerhalten, wenn der Algorithmus $\mathcal{A}'$ sein Ende erreicht hat. Zwischen m und m' besteht die klassische Beziehung von Hauptprogramm und Unterprogramm.

Um die Haupt-Unterprogramm-Beziehung zu realisieren, könnte der Algorithmus $\mathcal{A}$ einen Generierungspunkt

$$s_g = \left(n, \underline{\text{gen}}\left(\mathcal{A}', I(m')\right)\right) \in \mathcal{A} \tag{18}$$

enthalten, in dem die Maschine m eine neue Maschine generiert und aktiviert, die nach dem Algorithmus $\mathcal{A}'$ arbeiten soll. Die Rückgabe der Steuerung von m' an m erfolgt dann im Endpunkt

$$s^+_{\mathcal{A}'} = \left(n_e\ ,\ \underline{\text{gret}^+}\right) \in \mathcal{A}' \tag{19}$$

In diesem Endpunkt gibt die Maschine m' das Endergebnis gemeinsam mit der Steuerung an ihren Generator - also an die Maschine m - zurück. Gleichzeitig wird m' aus dem KI-Assistenten entfernt. Da die Maschine m nach der Generierung von m' erst dann die Steuerung zurückerhält, wenn m' schon nicht mehr existiert, braucht das erweiterte Steuerprogramm der Maschine m den Identifikator $I(m')$ gar nicht mitzuteilen. Wir benötigen also den Generierungspunkt nicht in der allgemeinen Form (18), sondern führen einen speziellen Generierungspunkt ein, den wir als ***Unterprogramm-Aufrufpunkt*** bezeichnen:

$$s_{ua} = \left(n\ ,\ \underline{\text{exec}}\left(\mathcal{A}'\right)\right) \in \mathcal{A} \tag{20}$$

Einen Algorithmus, der als einzigen Rückkehrpunkt einen Endpunkt gemäß (19) enthält und von einer Maschine ausgeführt wird, die in einem Unterprogramm-Aufrufpunkt generiert und aktiviert wurde, ordnen wir dem Typ ***Routine*** zu.

In der Abbildung 7 wird das Zeitverhalten beim Aufruf einer Routine als UML-Sequenzdiagramm dargestellt.

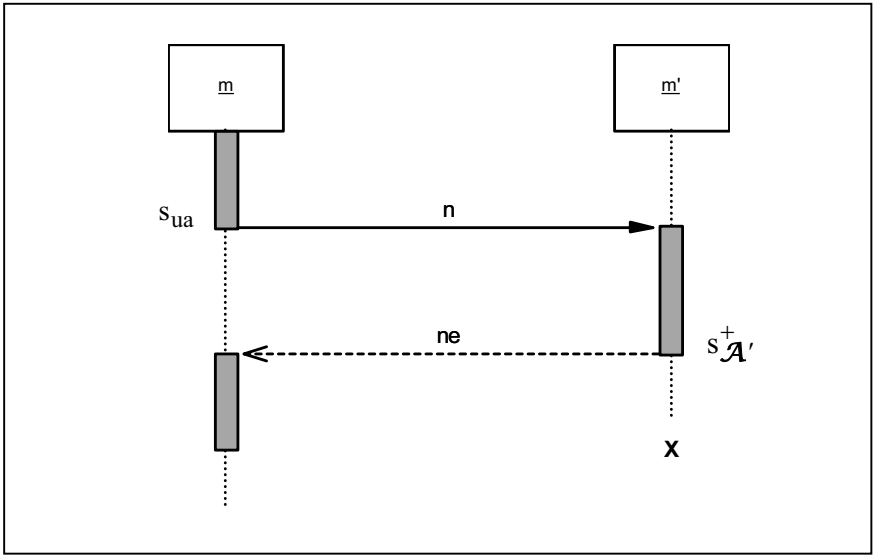


Abbildung 7: Zeitverhalten beim Aufruf einer Routine

Wie das Sequenzdiagramm zeigt, generiert die Maschine m in einem Unterprogramm-Aufrufpunkt s_{ua} die Maschine m' und übergibt ihr einen Auftrag in Form der Nachricht n. Die aufgerufene Maschine m' erreicht bei der Abarbeitung des Auftrags ihren Endpunkt $s^{+}_{\mathcal{A}'}$, beendet ihr Leben und gibt die Steuerung gemeinsam mit dem Ergebnis ne an ihren Generator m zurück.

Eine Maschine, die nach einem Algorithmus vom Typ einer Routine arbeitet, löst die ihr übertragene Aufgabe aus der Sicht ihres Generators in einem Schritt. Mitunter ist es für eine Maschine m jedoch rationeller, eine Maschine m' nacheinander jeweils mit der Lösung von Teilaufgaben zu betrauen und die bereitgestellten Teilergebnisse einzeln entgegenzunehmen.

Das Modell der abstrakten Maschinen ermöglicht diese Arbeitsweise. Die Maschine $m = (\mathcal{A}, d)$ generiert und aktiviert dazu in einem Generierungspunkt

$$s_g = (n_p, \underline{gen}(\mathcal{A}', I(m'))) \in \mathcal{A} \tag{21}$$

die neue Maschine $m' = (\mathcal{A}', d')$. Dabei wird der generierenden Maschine m der Identifikator $I(m')$ der neuen Maschine mitgeteilt. Die Nachricht n_p enthält im allgemeinen die Generierungsparameter. Die Maschine m' führt Initialisierungsaktivitäten aus. Sie erzeugt das Datenobjekt d' und belegt es mit Anfangswerten. Dann gibt sie die Steuerung in einem Aktivator-Rückkehrpunkt

$$s_{ar} = (n_0, \underline{aret}) \in \mathcal{A}' \tag{22}$$

gemeinsam mit der Nachricht n_0, die über den Erfolg der Generierung Auskunft gibt, an ihren Aktivator zurück, der in diesem Fall zugleich ihr Generator ist. Die Maschine m kann den Identifikator $I(m')$ nun dazu verwenden, m' nacheinander mit der Lösung von Teilaufgaben zu betrauen. Die Maschine m kann $I(m')$ in Sendepunkten auch anderen Maschinen des KI-Assistenten mitteilen und diesen damit die Möglichkeit geben, die Maschine m' ihrerseits mit der Lösung von Teilaufgaben zu beauftragen. Bei jeder Aktivierung von m' muss der jeweilige Algorithmus $\mathcal{A}^*$ des Aktivators einen Aktivierungspunkt

$$s_a = (n_a, \underline{act}(I(m'))) \in \mathcal{A}^* \tag{23}$$

enthalten, in dem m' die zu lösende Teilaufgabe in Form der Nachricht n_a mitgeteilt wird. Bei jeder solchen Aktivierung arbeitet m' solange, bis der

Algorithmus $\mathcal{A}'$ den nächsten Aktivator-Rückkehrpunkt erreicht. In diesem Sendepunkt schickt m' ein Teilergebnis n_t an seinen jeweiligen Aktivator zurück:

$$s_{ar} = (n_t, \underline{aret}) \in \mathcal{A}' \tag{24}$$

Nach Rückgabe der Steuerung an den Aktivator verharrt die Maschine m' im zuletzt erreichten Zustand, wobei ihr Datenobjekt d' unverändert erhalten bleibt. Bei der nächsten Aktivierung von m' setzt der Algorithmus $\mathcal{A}'$ seine Arbeit im Anschluss an den Aktivator-Rückkehrpunkt fort. Dabei kann er auf die unverändert gebliebenen Werte des Datenobjekts d' zurückgreifen.

Die Aktivierung der Maschine m' kann solange erfolgen, wie der Algorithmus $\mathcal{A}'$ noch nicht seinen Endpunkt erreicht hat. Im Modell der abstrakten Maschinen wurden hinsichtlich der Steuerungsübergabe im Endpunkt drei Möglichkeiten unterschieden: die Rückgabe an den Generator, die Rückgabe an den Aktivator und die Weitergabe an eine andere Maschine. Bei der Realisierung des Modells beschränken wir uns jedoch auf die Steuerungsrückgabe an den Generator, wie das beispielsweise auch in der Programmiersprache SIMULA[1] der Fall ist. Im Endpunkt, der nun also stets die Form

$$s^{+}_{\mathcal{A}'} = \left(n_e, \underline{gret^{+}}\right) \in \mathcal{A}' \tag{25}$$

hat, wird die Maschine m' aus dem System entfernt und das Endergebnis n_e gemeinsam mit der Steuerung dem Generator m übergeben.

Einen Algorithmus, der außer einem Endpunkt gemäß (25) noch wenigstens einen Aktivator-Rückkehrpunkt (24) besitzt, ordnen wir dem Typ ***Prozess*** zu.

Ein typischer „Lebenslauf“ einer Maschine m', die nach einem Algorithmus vom Typ eines Prozesses arbeitet, ist in der Abbildung 8 dargestellt.

[1] Vgl. beispielsweise Kirkerud (1989) und Sebesta (1996).

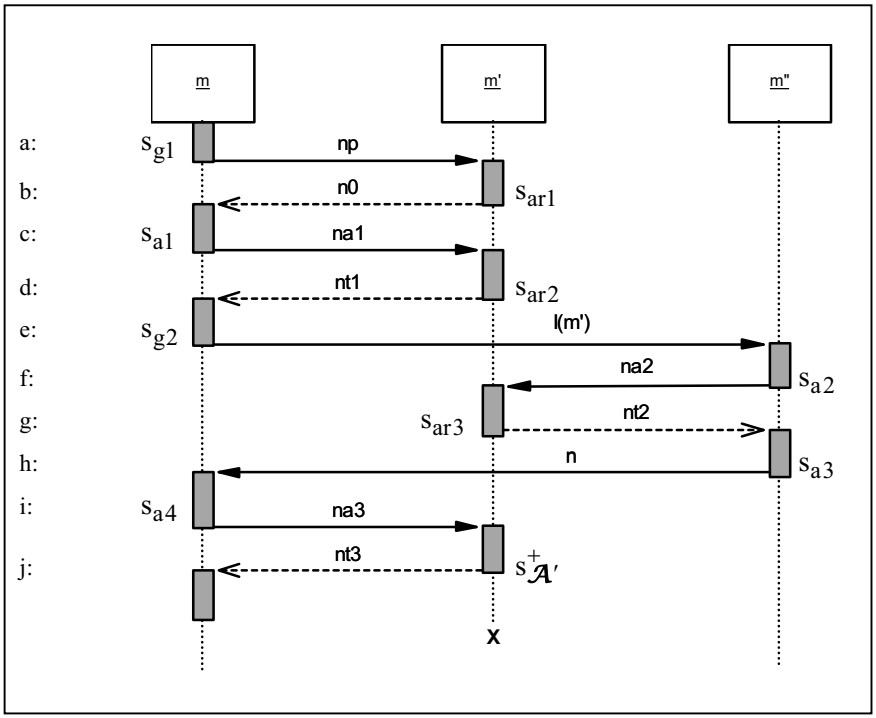


Abbildung 8: Zeitverhalten bei der wiederholten Aktivierung eines Prozesses

Im Zeitverlauf werden nacheinander die folgenden Sendepunkte durchlaufen:

a: Im Generierungspunkt s_{g1} generiert und aktiviert die Maschine m die neue Maschine m', die im Sinne eines Servers nacheinander Teilleistungen für verschiedene Client-Maschinen erbringen soll. Die Nachricht np enthält dabei die Generierungsparameter. Das erweiterte Steuerprogramm teilt dem Generator m den Identifikator $I(m')$ der Server-Maschine mit.

b: Die Server-Maschine m' hat ihre Initialisierungsoperationen abgeschlossen und gibt im Aktivator-Rückkehrpunkt s_{ar1} die Steuerung und die Nachricht $n0$ an ihren Generator zurück. Sie steht jetzt bereit, um für jede

Maschine, die ihren Identifikator kennt, die jeweils angeforderte Dienstleistung zu erbringen.

c: Die Maschine m aktiviert im Sendepunkt s_{a1} die Server-Maschine m' durch Zusenden der Nachricht $na1$.

d: Die Server-Maschine m' hat die von der Client-Maschine m in Auftrag gegebene Teilleistung erbracht und gibt die Ergebnisse im Aktivator-Rückkehrpunkt s_{ar2} in Form der Nachricht $nt1$ zurück.

e: Bisher war lediglich die Maschine m als der Generator der Server-Maschine m' im Besitz des Identifikators $I(m')$. Im Generierungspunkt s_{g2} ruft m eine weitere Maschine m'' ins Leben und teilt ihr dabei den Identifikator der Server-Maschine mit.

f: Die Maschine m'' kann nun im Aktivierungspunkt s_{a2} die Server-Maschine in Form der Nachricht $na2$ damit beauftragen, eine weitere Teilleistung zu erbringen.

g: Die Ergebnisse dieser Teilleistung sendet die Server-Maschine m' im Aktivator-Rückkehrpunkt s_{ar3} als Nachricht $nt2$ an ihren aktuellen Auftraggeber m'' zurück.

h: Im Aktivierungspunkt s_{a3} übergibt m'' gemeinsam mit der Nachricht n die Steuerung an die Maschine m.

i: Die Maschine m übergibt im Sendepunkt s_{a4} der Server-Maschine m' einen weiteren Auftrag in Form der Nachricht $na3$.

j: Die Server-Maschine m' hat ihren Endpunkt $s^+_{\mathcal{A}'}$ erreicht, gibt das Ergebnis ihrer letzten Teilleistung als Nachricht $nt3$ an ihren Generator m zurück und beendet ihr Leben.

3.4 Typische Teilsysteme abstrakter Maschinen

Wir betrachten in diesem Abschnitt typische Formen des Zusammenwirkens von abstrakten Maschinen und geben an, wie sie im Rahmen unseres Modells verwirklicht werden können.

Werden mehrere Maschinen generiert, um gemeinsam eine komplexe Aufgabe zu lösen, und treten sie dabei zeitweilig miteinander in eine intensive Wechselwirkung, so bilden sie innerhalb des Systems $\mathcal{M}^{\mathrm{t}}$ ein ***Teilsystem abstrakter Maschinen***. Beim Entwurf von KI-Assistenten treten einige Typen solcher Teilsysteme besonders häufig auf. Diese wollen wir im folgenden näher betrachten.

Hierarchische Systeme von Maschinen, die zueinander in der Haupt-Unterprogramm-Beziehung stehen, lassen sich durch Algorithmen vom Typ einer Routine in einfacher Weise aufbauen. Auf dieses allgemein bekannte Prinzip wollen wir nicht näher eingehen.

Komplexere Teilsysteme von Maschinen lassen sich durch Algorithmen vom Typ eines Prozesses aufbauen. Besondere Bedeutung erlangen dabei Teilsysteme, die die Realisierung von Elementen der objektorientierten Programmierung - insbesondere von abstrakten Datentypen und von Koprogrammen - ermöglichen.

Wir wollen den Begriff des abstrakten Datentyps schrittweise einführen. Dazu erläutern wir zunächst das Zusammenwirken zweier Maschinen, von denen die eine die ihr zugedachte Gesamtaufgabe löst, indem sie eine Folge von Teilergebnissen liefert. Dann erweitern wir das Maschinensystem auf den Fall, dass mehrere Maschinen diese Teilergebnisse abfordern. Schließlich werden wir beschreiben, wie mehrere Maschinen, die nach demselben Algorithmus arbeiten, Aufgaben in Teilschritten lösen und damit die Exemplare eines abstrakten Datentyps bilden.

3.4.1 Lösung einer Aufgabe in Teilschritten

Der KI-Assistent besteht aus einem Ensemble miteinander wechselwirkender abstrakter Maschinen. Eine dieser Maschinen sei $\mathrm{m} = (\mathcal{A}, \mathrm{d})$. Ihr Algorithmus $\mathcal{A}$ enthalte die folgende Teilmenge von Sendepunkten:

$$\mathcal{A} \supseteq \{ s_g \} \cup \mathcal{A}^*$$
$$\mathcal{A}^* = \{ s_{\mathcal{A}1}, s_{\mathcal{A}2}, \ldots, s_{\mathcal{A}n} \} \tag{26}$$

In diesen Sendepunkten soll die Zusammenarbeit von m mit einer anderen Maschine m' erfolgen. Darüber hinaus kann $\mathcal{A}$ weitere Sendepunkte enthalten, die für diese Zusammenarbeit jedoch unwesentlich sind. Wenn die Maschine m den Generierungspunkt $s_g = (n_g, \underline{\text{gen}}(\mathcal{A}', I(m')))$ erreicht, fordert sie das erweiterte Steuerprogramm auf, die nach dem Algorithmus $\mathcal{A}'$ arbeitende Maschine m' zu generieren und zu aktivieren. Wie bereits im Abschnitt 3.3 erläutert wurde, führt m' mit dem Datenobjekt Initialisierungsoperationen aus und gibt zum angenommenen Zeitpunkt t_0 im Aktivator-Rückkehrpunkt $s'^{t_0}_{ar} = (n_0, \underline{\text{aret}}) \in \mathcal{A}'$ die Steuerung an m zurück. Nun ist die Maschine m' zur Lösung der ihr zugedachten Gesamtaufgabe in Teilschritten eingerichtet. Diese Aufgabe wird hinsichtlich der Aufgabenklasse durch den Algorithmus $\mathcal{A}'$ festgelegt und durch die Generierungsparameter n_g genauer spezifiziert.

In der weiteren Abarbeitung des Algorithmus $\mathcal{A}$ erreicht die Maschine m zu einem Zeitpunkt t einen Aktivierungspunkt $s^t_{\mathcal{A}i} = (n_t, \underline{\text{act}}(I(m'))) \in \mathcal{A}^*$, in dem sie die Maschine m' zum Zwecke der Ausführung eines Lösungsschritts aktiviert. Die einzelnen Aktivierungspunkte von $\mathcal{A}^*$ können in spezifischer Folge durchlaufen werden. Auf Grund von Zyklen und Rekursivitäten kann ein gegebener Aktivierungspunkt $s_{\mathcal{A}i} \in \mathcal{A}^*$ auch mehrfach erreicht werden.

Der nach einer Aktivierung von m' ausgeführte Informationsverarbeitungsprozess hängt in seinem Verlauf von vier Faktoren ab:

1. vom Bearbeitungszustand, den der Algorithmus $\mathcal{A}'$ im zuletzt erreichten Aktivator-Rückkehrpunkt erreicht hat,

2. vom Zustand des Datenobjekts d' in diesem Aktivator-Rückkehrpunkt,

3. von den Aktivierungsparametern n_t und

4. von den Nachrichten, die m' durch eine eventuelle Kommunikation mit dem Benutzer oder mit anderen Maschinen des Systems $\mathcal{M}^t$ - beispielsweise mit der Maschine 'IRS' - erhält.

Der Lösungsschritt ist beendet, wenn m' den nächsten Rückkehrpunkt erreicht. Sie übergibt dann ein Teilergebnis n'_t an die Maschine m. Dieser Rückkehrpunkt ist entweder ein Aktivator-Rückkehrpunkt $s'^{t}_{ar} = (n'_t, \underline{aret})$ oder der Endpunkt des Algorithmus $\mathcal{A}'$: $s^{+}_{\mathcal{A}'} = (n'_t, \underline{gret^{+}})$. Mit dem Erreichen des Endpunktes hat m' die Gesamtaufgabe erfüllt und wird durch das erweiterte Steuerprogramm aus dem KI-Assistenten entfernt.

Das Zusammenspiel der Maschinen m und m' wird als UML-Sequenzdiagramm in der Abbildung 9 veranschaulicht.

3.4.2 Abfordern von Teilergebnissen durch mehrere Maschinen

Wir wollen nun zu dem anspruchsvolleren Fall übergehen, dass die von der Maschine m' zu liefernden Teilergebnisse nicht vom Generator der Maschine m' abgefordert werden, sondern von anderen Maschinen des KI-Assistenten.

Eine Maschine m fungiert als Generator und richtet beim Erreichen eines Generierungspunktes s_g die Maschine m' ein. Nun steht m' zur Verfügung, und die Maschine m kennt den Identifikator $I(m')$. Diesen Identifikator übergibt m in Form einer Nachricht n_I in Sendepunkten s_{Ii} an die Maschinen $m_i = (\mathcal{A}_i, d_i)$; $i = 1,2,\ldots,k$. Diese Maschinen speichern den Identifikator jeweils in ihrem Datenobjekt d_i.

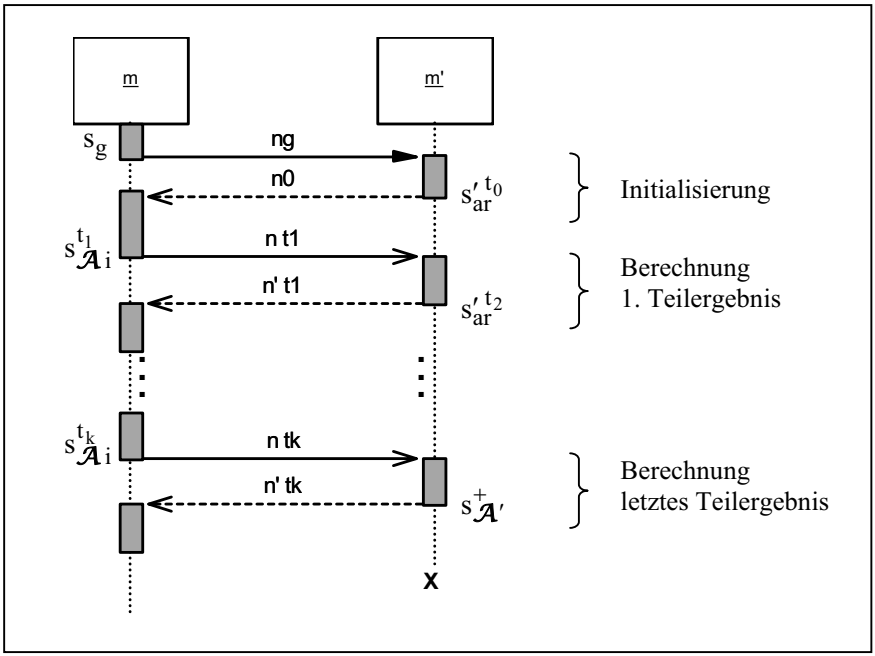


Abbildung 9: Zusammenspiel der Maschinen m und m' bei der Lösung einer Aufgabe in Teilschritten

Jede der Maschinen m_i ; $i = 1,2,\ldots,k$ kann nun in einem Aktivierungspunkt $s^t_a = (n_t, \underline{act}(I(m'))) \in \mathcal{A}_i$ die Maschine m' aktivieren und so ein Teilergebnis von ihr abfordern. Die Übergabe des Teilergebnisses erfolgt jeweils in einem Aktivator-Rückkehrpunkt $s^t_{ar} = \left(n^t_e, \underline{aret}\right) \in \mathcal{A}'$. Hier nun zeigt sich die Zweckmäßigkeit der Relation α^t, durch deren Auswertung n^t_e an den jeweiligen Aktivator von m' übergeben werden kann.

Da mehrere Maschinen Teilergebnisse von m' abfordern, muss die zeitliche Aufeinanderfolge der einzelnen Aktivierungen koordiniert werden. Zu diesem Zweck müssen die Maschinen m_i; $i = 1,2,\ldots,k$ einander die Steuerung übergeben und Informationen über den erreichten Verarbeitungsstand von m' austauschen.

Ein einfacher und häufig auftretender Fall liegt dann vor, wenn die Maschine m' die Aufgabe hat, einzelne Dienstleistungen für unterschiedliche Auftraggeber auszuführen. Soll lediglich aus den Daten des Auftraggebers ein Ergebnis berechnet werden, ohne dass im Datenobjekt d' Verarbeitungszustände „gemerkt" werden müssen, so kann $\mathcal{A}'$ vom Typ einer Routine sein. Ein Beispiel dafür ist die Berechnung einer Winkelfunktion. Benötigt jedoch m' für seine Arbeit ein „Gedächtnis", das auch noch nach der Rückgabe der Steuerung erhalten bleiben soll, so muss $\mathcal{A}'$ als Prozess gestaltet werden. Ein Beispiel dafür ist eine Maschine, die einen Speicherbereich verwaltet und die als Dienstleistungen Speicherabschnitte bereitstellt und sie wieder entgegennimmt, wenn diese nicht mehr benötigt werden. Zur Verwaltung des Speicherbereichs benötigt die Maschine m' ein Datenobjekt d', das sowohl den Speicherbereich als auch die Informationen über dessen Verbrauch enthält. Die einzelnen Dienstleistungen sind voneinander unabhängig, so dass die einzelnen Aktivierungen von m' nicht koordiniert werden müssen. Im Algorithmus $\mathcal{A}'$ hat das zur Folge, dass die Rückgabe der Steuerung an den Aktivator zweckmäßigerweise stets im selben Aktivator-Rückkehrpunkt s'_{ar} erfolgt. Der Algorithmus $\mathcal{A}'$ muss dabei so gestaltet werden, dass die Maschine m' solange nicht ihren Endpunkt $s^{+}_{\mathcal{A}'}$ erreicht, wie ihre Dienstleistungen noch benötigt werden. In einer speziellen Aktivierung wird m' schließlich beauftragt, die erforderliche Endorganisation vorzunehmen und dann die Arbeit zu beenden. Aus Gründen einer übersichtlichen Systemarchitektur sollte diese letzte Aktivierung von m' in einem Aktivierungspunkt $s^{*}_{a} \in \mathcal{A}$ durch diejenige Maschine erfolgen, die m' auch eingerichtet hat, also durch den Generator m. Diese Form des Zusammenspiels von Maschinen wird in Abbildung 10 veranschaulicht. Die gerichteten Strich-Punkt-Linien symbolisieren dabei eine Steuerungsübergabe, deren konkrete Form in diesem Zusammenhang unwesentlich ist.

3.4.3 Exemplare abstrakter Datentypen

Zuletzt haben wir die Situation beschrieben, dass eine Maschine $m' = (\mathcal{A}', d')$ von mehreren anderen Maschinen zu dem Zweck aktiviert wird, einzelne Teilergebnisse bereitzustellen. Wir nutzen nun die Möglichkeit unseres Modells, mehrere Maschinen zu generieren, die zwar alle nach demselben Algorithmus ar-

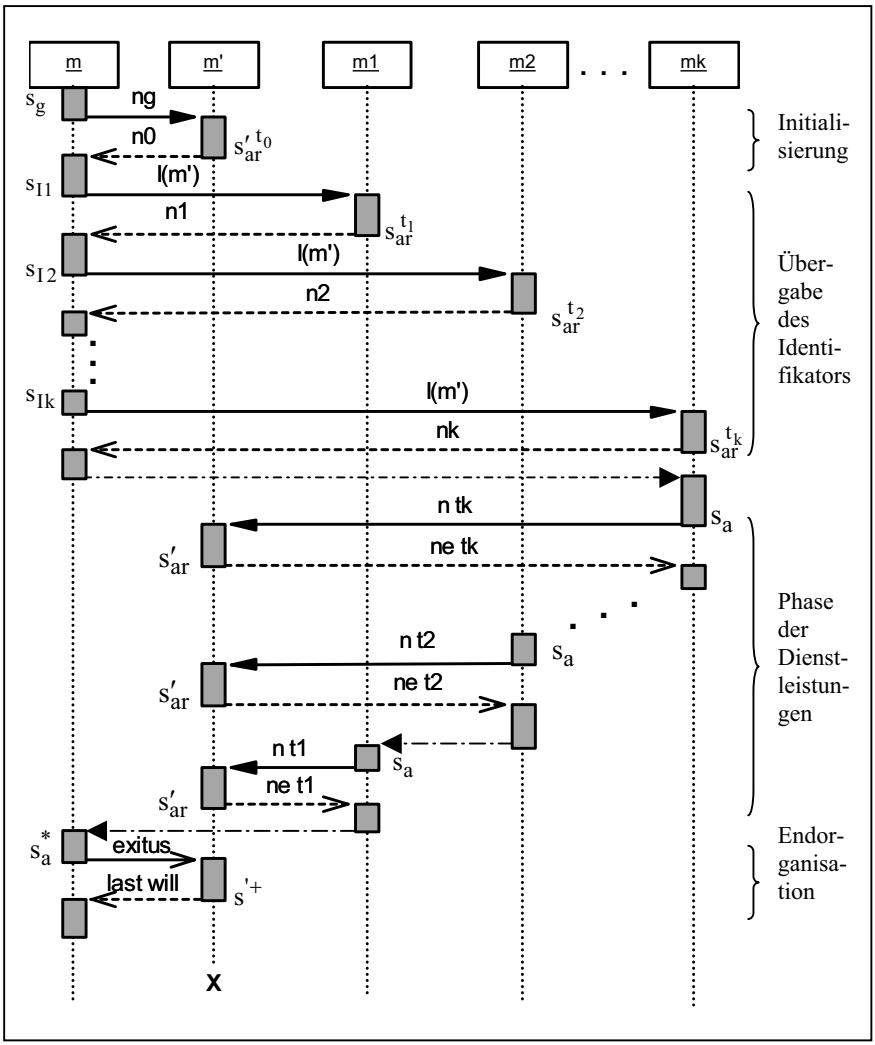


Abbildung 10: Zusammenspiel von Maschinen beim Abfordern von Dienstleistungen

beiten, von denen aber jede ein individuelles Datenobjekt besitzt und somit ein Exemplar eines abstrakten Datentyps darstellt.

Mit diesem Vorgehen wollen wir erreichen, dass nur an einer Stelle - nämlich allein im Algorithmus $\mathcal{A}'$ - die konkrete Struktur der Daten und die mit ihnen ausführbaren Operationen beschrieben werden müssen. In allen anderen Maschinen braucht dann die Struktur der Daten nicht mehr bekannt zu sein. Sie müssen lediglich wissen, welche Operationen ausführbar sind. Dabei ist ihnen wiederum unbekannt, nach welchem Algorithmus die Operationen vollzogen werden. Wenn wir jetzt noch zulassen, dass es mehrere Exemplare solcher Daten gibt, die mit demselben Satz von Operationen manipuliert werden können, so gelangen wir zum Konzept des abstrakten Datentyps[1], das ein wesentliches Element der objektorientierten Programmierung bildet.[2]

Ein abstrakter Datentyp T wird in unserem Modell wie folgt realisiert. Sowohl die Struktur der Daten als auch die über ihnen erklärten Operationen werden in einem Algorithmus $\mathcal{A}_T$ beschrieben. Wir nehmen an, dass von einer abstrakten Maschine $m = (\mathcal{A}, d)$ n verschiedene Exemplare des abstrakten Datentyps T benötigt werden. Die Maschine m generiert dann diese Exemplare in den Generierungspunkten

$$s_{gi} = \left(n_i, \underline{gen}\left(\mathcal{A}_T, I(m_i)\right)\right) \in \mathcal{A}; \quad i = 1,2,...,n \tag{27}$$

in Gestalt der Maschinen $m_i = (\mathcal{A}_T, d_i)$; $i = 1,2,\ldots,n$. Dabei enthält die Nachricht n_i die Parameter zum Einrichten des individuellen Datenobjekts d_i. Im Anschluss an die Initialisierung von d_i gibt die Maschine m_i die Steuerung an m zurück. Die Maschine m kennt den Identifikator $I(m_i)$ und kann ihn anderen Maschinen mitteilen. Jedes Exemplar des abstrakten Datentyps hat seinen individuellen Identifikator. Dadurch besteht nun die Möglichkeit, dass eine beliebige Maschine das jeweils gewünschte Exemplar des abstrakten Datentyps durch $I(m^*)$ auswählt und aktiviert:

---

[1] Vgl. beispielsweise Zoller (1981) und Sommerville (2001), S. 641.
[2] Vgl. beispielhaft Brügge/Dutoit (2004).

$$s_a = \left(n_{op}, \underline{act}\left(I\left(m^*\right)\right)\right) \tag{28}$$

Die Nachricht n_{op} beinhaltet die Angabe, welche Operation das Exemplar ausführen soll. Nach Ausführung der Operation gibt das Exemplar gemeinsam mit der Steuerung in einem Aktivator-Rückkehrpunkt das Ergebnis der Operation an den Aktivator zurück.

Bei jeder Aktivierung kann sich das jeweilige Exemplar des abstrakten Datentyps ändern. Selbst dann, wenn alle Exemplare mit den gleichen Parameterwerten generiert wurden, unterscheiden sie sich doch recht bald auf Grund der individuell unterschiedlichen Aktivierungen.

3.4.4 Die Maschine 'IRS'

Das System der abstrakten Maschinen, das den KI-Assistenten bildet, enthält eine spezielle Maschine $'IRS' = (STAIRS, d)$, die nach dem Algorithmus des Information-Retrieval-Systems STAIRS ein Datenobjekt d bearbeitet. Das Datenobjekt d bildet das „Gedächtnis“ des Information-Retrieval-Systems und enthält alle Daten, die STAIRS für die Ausführung seiner Dienstleistungen benötigt.

Die Maschine $'IRS'$ arbeitet nach einem Algorithmus vom Typ eines Prozesses. Sie kann von allen Maschinen des KI-Assistenten aktiviert werden. Auf Grund ihrer exponierten Stellung im System $\mathcal{M}^t$ weist sie eine Reihe von Besonderheiten auf:

1. Die Maschine $'IRS'$ kann durch keine andere Maschine des Systems $\mathcal{M}^t$ generiert werden. Sie wurde gemeinsam mit der Monitormaschine zum Zeitpunkt t_0 gemäß (4) eingerichtet.

2. Ihr Identifikator $I('IRS')$ steht allen Maschinen des KI-Assistenten a priori zur Verfügung.

3. Wird die Maschine 'IRS' von einer anderen Maschine des KI-Assistenten in einem Aktivierungspunkt $s_a = (n, \underline{act}(I('IRS')))$ zur Fortsetzung ihrer Arbeit aufgefordert, so muss die gesendete Nachricht n ein gültiger Ausdruck gemäß der Anfragesprache von STAIRS sein. Nun ist aber in einem konkreten Verarbeitungszustand des Information-Retrieval-Systems nicht jedes Kommando zulässig. Der Aktivator muss daher entweder den jeweils erreichten Verarbeitungsstand von STAIRS zuverlässig kennen oder er muss zuvor eines derjenigen Kommandos senden, die stets erlaubt sind und STAIRS in einen definierten Zustand versetzen.

4. Die Nachrichten, die die Maschine 'IRS' in den Aktivator-Rückkehrpunkten an den jeweiligen Aktivator zurückgibt, gehören zur Menge der Ausgabenachrichten von STAIRS. Da sie eigentlich für den Benutzer bestimmt sind, enthalten sie in der Regel unformatierte Texte.

5. Das Information-Retrieval-System STAIRS soll auch dann noch weiterarbeiten können, wenn der KI-Assistent wieder abgebaut wurde. Demzufolge darf die Maschine 'IRS' nicht mit der Nachricht "..OFF" aktiviert werden, da STAIRS auf dieses Kommando hin seine Arbeit einstellt.

3.4.5 Teilsysteme von Komaschinen

In den bisher beschriebenen Teilsystemen hat eine Maschine m', die von einem Aktivator m aktiviert wurde, in einem Aktivator-Rückkehrpunkt die Steuerung stets wieder an ihren Aktivator m zurückgegeben. Dabei wurde die Relation α^t ausgenutzt, die mit dem Element $(m, m') \in \alpha^t$ eine hierarchische Abhängigkeit zwischen m und m' ausweist. Wir betrachten nun ein Teilsystem von Maschinen, bei dem die Übergabe der Steuerung während eines Zeitintervalls $[t_1, t_2]$ ausschließlich in Aktivierungspunkten erfolgt. Bei jeder dieser Aktivierungen wird zwar weiterhin die Relation α^t gemäß (10) aktualisiert, doch bleiben diese hierarchischen Abhängigkeiten bewusst ungenutzt: Alle Maschinen des Teilsystems sind gleichberechtigt - jede Maschine kann jede andere aktivieren.

Ein Teilsystem von abstrakten Maschinen, bei dem die Steuerungsübergabe während eines Zeitintervalls $[t_1, t_2]$ ausschließlich in Aktivierungspunkten erfolgt, bezeichnen wir als ein ***Teilsystem von Komaschinen***.[1] Dieses Teilsystem ist erst ab dem Zeitpunkt t_1 arbeitsfähig. Dann sind alle seine Maschinen generiert worden, und jede dieser Maschinen „kennt" die Identifikatoren sämtlicher anderer Maschinen.

Wir betrachten die Phasen des Aufbaus, der Arbeit und des Abbaus eines Teilsystems von Komaschinen an Hand eines Beispiels.

Eine Maschine m generiert zwei weitere Maschinen m_1 und m_2, teilt m_1 den Identifikator $I(m_2)$ und m_2 den Identifikator $I(m_1)$ mit. Das aus den beiden Maschinen m_1 und m_2 bestehende Teilsystem von Komaschinen kann nun seine Arbeit aufnehmen.

Dazu aktiviert die Maschine m zum Teilpunkt t_1 beispielsweise die Maschine m_1. Im weiteren Verlauf möge m_1 jeweils m_2 und m_2 jeweils m_1 aktivieren. Das erfolgt solange, bis zum Zeitpunkt t_2 eine der Maschinen m_1 oder m_2 ihren Endpunkt erreicht.

Jetzt wird das Teilsystem der Komaschinen abgebaut. Die Maschine, die ihren Endpunkt erreicht hat - im Beispiel m_1 -, wird aus dem System $\mathcal{M}^{t_2}$ entfernt, wobei die Steuerung an ihren Generator zurückgegeben wird. Eine solche Rückgabe der Steuerung an den Generator ist bei Komaschinen zweckmäßig: Die Maschine m, die das Teilsystem von Komaschinen eingerichtet und aktiviert hat, erfährt somit, dass das Teilsystem seine Arbeit beendet hat, und kann auf dieses Ereignis reagieren.

Die Maschine m muss nun das Teilsystem abbauen. Eine der Maschinen - in unserem Fall m_1 - ist bereits gelöscht. Da eine Rückgabe der Steuerung stets ohne Angabe des „Absenders" erfolgt, muss m die Möglichkeit eingeräumt werden zu erfragen, ob sich eine in Frage stehende Maschine noch im KI-Assistenten befindet. Die Maschine m kann das erweiterte Steuerprogramm gegebe-

[1] Vgl. beispielsweise Knuth (1972).

nenfalls anweisen, eine noch existierende Maschine „gewaltsam“ zu löschen. Dabei wird das System $\mathcal{M}^t$ gemäß (16) und (17) genauso verändert, als hätte die Maschine von selbst ihren Endpunkt erreicht.

Der Steuerungsfluss des betrachteten Beispiels ist in Abbildung 11 dargestellt. Dabei kennzeichnet das Symbol $\oplus$ das Ereignis des „gewaltsamen“ Löschens einer Maschine.

3.5 Kommunikation des KI-Assistenten mit dem Benutzer

Bisher wurde lediglich beschrieben, wie in Sendepunkten Nachrichten zwischen den abstrakten Maschinen des Systems $\mathcal{M}^t$ ausgetauscht werden. Neben dieser internen Kommunikation innerhalb des Systems kann jedoch jede Maschine des KI-Assistenten auch einen Dialog mit dem Benutzer führen. Diese Dialoge realisieren die externe Kommunikation des Maschinensystems.

Die erste externe Kommunikation von $\mathcal{M}^t$ erfolgt in einem ***Eintrittspunkt***. Dieser Eintrittspunkt liegt am Beginn des Algorithmus, nach dem die Monitormaschine arbeitet. Wenn das erweiterte Steuerprogramm die Monitormaschine einrichtet, übergibt es ihr neben der Steuerung auch die erste Nachricht des Benutzers.

Während des Informationsverarbeitungsprozesses des KI-Assistenten können ***Dialogpunkte*** durchlaufen werden. Erreicht die Maschine $\mathrm{m} = (\mathcal{A}, \mathrm{d})$ einen solchen Dialogpunkt, so schickt sie dem Benutzer eine Nachricht zu und verharrt solange im Dialogpunkt, bis die Antwort des Benutzers eingetroffen ist. Die Antwort wird vom Algorithmus $\mathcal{A}$ ausgewertet; sie beeinflusst im allgemeinen die weitere Arbeit des KI-Assistenten.

Erkennt eine abstrakte Maschine, dass die dem KI-Assistenten übertragene Aufgabe vollständig gelöst wurde, so informiert sie den Benutzer darüber und veranlasst, dass das Maschinensystem $\mathcal{M}^t$ abgebaut wird. Einen Punkt, in dem dies geschieht, bezeichnen wir als einen ***Austrittspunkt***. Während es innerhalb eines KI-Assistenten nur ***einen*** Eintrittspunkt gibt, kann es ***mehrere*** Austrittspunkte

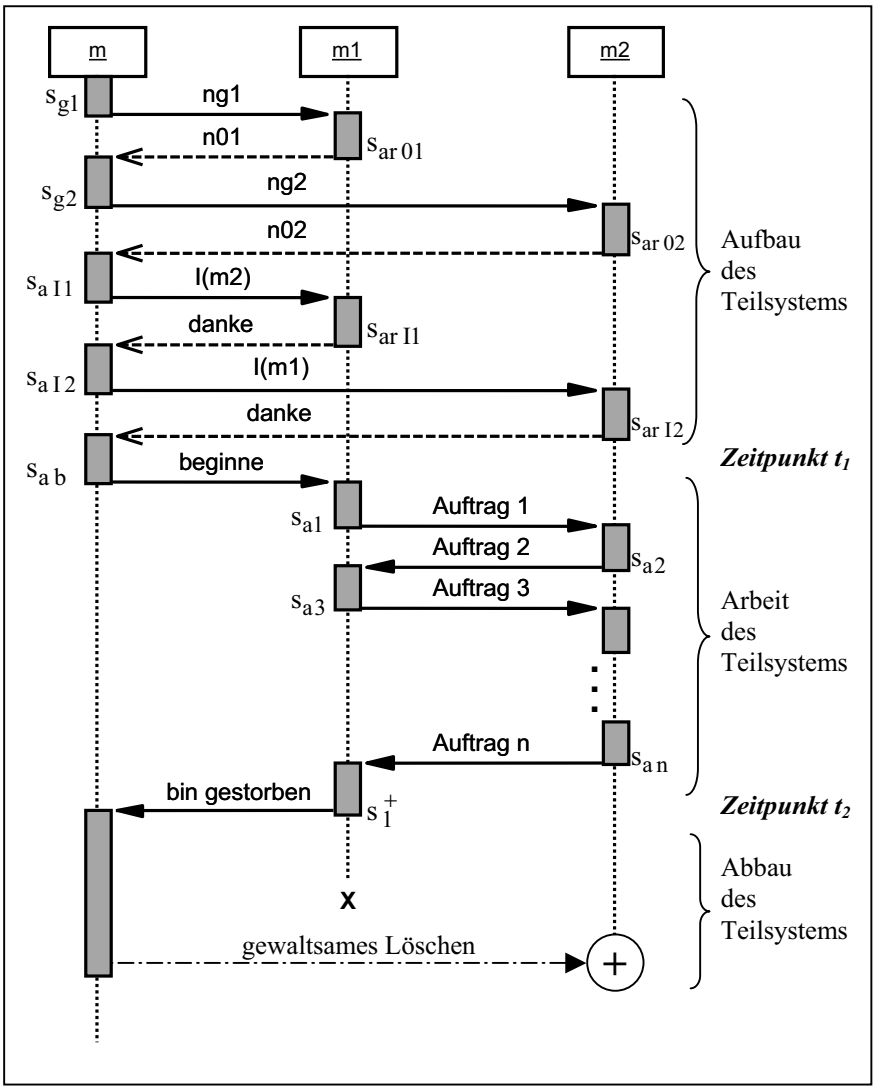


Abbildung 11: Aufbau, Arbeit und Abbau eines Teilsystems von Komaschinen

geben, die in den Algorithmen verschiedener Maschinen liegen können. Aus diesem Grunde sollte der Benutzer beim Abbau des KI-Assistenten darüber informiert werden, welche der möglichen Endsituationen eingetreten ist. Dazu wird dem Benutzer im Austrittspunkt eine letzte Nachricht des KI-Assistenten zugeschickt.

3.6 Entstehen und Vergehen des KI-Assistenten

Beginnt ein Benutzer seine Bildschirmsitzung mit dem Information-Retrieval-System STAIRS, so wurde noch kein KI-Assistent gestartet. STAIRS fordert den Benutzer jeweils zur Eingabe eines Kommandos auf, analysiert es, führt das Kommando aus und informiert den Benutzer über das Ergebnis. Diese Kommunikation erfolgt – wie im Abschnitt 3.1 dargestellt wurde - durch Vermittlung des Steuerprogramms.

Auf Grund der von uns vorgenommenen Erweiterungen des Steuerprogramms werden die Eingaben des Benutzers vom nunmehr erweiterten Steuerprogramm „beobachtet“. Solange die Eingaben an das Information-Retrieval-System gerichtet sind, werden sie einfach an STAIRS „durchgereicht“. Der Benutzer hat aber jederzeit die Möglichkeit, durch eine spezielle Anweisung an das erweiterte Steuerprogramm den Aufbau eines KI-Assistenten auszulösen. Diese Anweisung enthält zum einen den Namen des Algorithmus, nach dem die Monitormaschine arbeiten soll, und zum anderen die Nachricht, die der Monitormaschine im Eintrittspunkt zu übergeben ist.

Wie im Abschnitt 3.2 beschrieben wurde, richtet das erweiterte Steuerprogramm beim Aufbau eines KI-Assistenten das System $\mathcal{M}^{t_0} = (\{'\mathrm{IRS}', m_0\}, \varnothing, \varnothing)$ ein, das außer der Maschine $'\mathrm{IRS}'$ auch noch die Monitormaschine m_0 enthält.

Das Information-Retrieval-System STAIRS besitzt in seiner ursprünglichen Fassung natürlich keine Sendepunkte und ist daher auch keine abstrakte Maschine im Sinne unseres Modells. STAIRS enthält jedoch Verarbeitungspunkte, in denen es Kommandos vom Benutzer erwartet und zu diesem Zweck die Steuerung an das erweiterte Steuerprogramm abtritt. Das Information-Retrieval-System STAIRS wird nun dadurch zur abstrakten Maschine $'\mathrm{IRS}'$, dass diese Verarbeitungspunkte zu Aktivator-Rückkehrpunkten „umfunktioniert“ werden.

Zum Zeitpunkt t_0 beginnt das Monitorprogramm mit seiner Arbeit. Dabei laufen im KI-Assistenten die bereits ausführlich beschriebenen Veränderungen ab. Schließlich tritt irgendwann die Situation ein, dass eine Maschine des Systems einen Austrittspunkt erreicht. Dann löscht das erweiterte Steuerprogramm alle noch im KI-Assistenten vorhandenen Maschinen. Die Aktivator-Rückkehrpunkte der Maschine 'IRS' werden wieder als ursprüngliche Verarbeitungspunkte des Information-Retrieval-Systems STAIRS betrachtet. Dem Benutzer wird eine letzte Nachricht des KI-Assistenten zugeschickt. Diese informiert ihn über den erreichten Austrittspunkt und fordert ihn auf, die ursprüngliche Kommunikation mit dem Information-Retrieval-System wieder aufzunehmen.

Der Benutzer hat danach allerdings jederzeit wieder die Möglichkeit, das erweiterte Steuerprogramm anzuweisen, denselben oder einen anderen KI-Assistenten einzurichten.

3.7 Formen der Kommunikation

In diesem Kapitel wurde ein geschlossenes Modell für die Mensch-Maschine-Kommunikation bei der Nutzung eines Information-Retrieval-Systems entwickelt. Während bei der ursprünglichen Systemkonfiguration das IRS als alleiniger maschineller Kommunikationspartner des Benutzers fungierte, stellt das beschriebene Modell dem IRS die abstrakten Maschinen des KI-Assistenten als weitere maschinelle Kommunikationspartner zur Seite. Um ein qualitativ höheres Niveau der Mensch-Maschine-Kommunikation zu erreichen, findet die Kommunikation nun auf drei Ebenen statt:

1. Kommunikation des Benutzers mit den abstrakten Maschinen des KI-Assistenten. Die abstrakten Maschinen verkörpern in ihrem Zusammenspiel die zusätzliche Intelligenz, so dass der Benutzer einen intelligenteren Kommunikationspartner gewinnt, als es das IRS allein gewesen ist.

2. Kommunikation der abstrakten Maschinen des KI-Assistenten mit dem IRS. Durch diese Kommunikation wird der programmierte Dialog realisiert. Die abstrakten Maschinen nutzen das IRS als eine komfortable „Zugriffsmethode“, durch die sie auf bibliographische Informationen zugreifen können, indem sie - entsprechend dem deklarativen Paradigma[1] - lediglich die

[1] Vgl. beispielsweise Stahlknecht/Hasenkamp (2005), S. 286.

Eigenschaften der gesuchten Informationen angeben. Auf dieser Ebene der Kommunikation findet die Massendatenverarbeitung statt, von der die Mensch-Maschine-Kommunikation nun entlastet wird.

3. Kommunikation der abstrakten Maschinen des KI-Assistenten untereinander. Das Niveau dieser Kommunikation ist durch die ereignisabhängige Steuerungs- und Nachrichtenübergabe gekennzeichnet. Eine abstrakte Maschine des KI-Assistenten wird dadurch zu einem „anspruchsvollen" Kommunikationspartner, dass sie über ein eigenes „Gedächtnis" verfügt, einen individuellen Stand ihrer Informationsverarbeitung hat und zu beliebigen Zeitpunkten in Kommunikation mit anderen abstrakten Maschinen treten kann. „Ebenbürtige" Kommunikationspartner entstehen durch das Generieren mehrerer Maschinen, die nach demselben Algorithmus arbeiten und damit die Exemplare eines abstrakten Datentyps bilden.

4 Programmiersprache für den KI-Assistenten

In diesem Kapitel stellen wir uns die Aufgabe, eine Programmiersprache zu entwickeln, die die Ausdrucksmittel zur Realisierung unseres Modells der Mensch-Maschine-Kommunikation bereitstellt. Das Niveau dieser Programmiersprache soll durch folgende qualitative Faktoren gekennzeichnet sein:

1. Sie muss die Anwendungsprogrammierung auf dem Niveau einer höheren Programmiersprache ermöglichen,

2. Sie muss die Kommunikation auf drei Ebenen unterstützen:

   2.1. Kommunikation des KI-Assistenten mit dem Benutzer,

   2.2. Kommunikation des KI-Assistenten mit dem Information-Retrieval-System und

   2.3. Kommunikation der abstrakten Maschinen des KI-Assistenten miteinander.

3. Sie muss einen ereignisabhängigen Auf- und Abbau des KI-Assistenten ermöglichen und die Arbeit mit abstrakten Datentypen und Koprogrammen gewährleisten.

Die Algorithmen, nach denen die Maschinen des KI-Assistenten arbeiten sollen, werden in Anwenderprogrammen formuliert. Für das Formulieren der Anwenderprogramme wurde die Programmiersprache KOMPROMISS[1] entwickelt, die PL/I als Wirtssprache verwendet.

Wir folgen dem Grundgedanken des Modells der abstrakten Maschinen und stellen zunächst die sprachlichen Konstruktionen vor, mit denen in KOMPROMISS Sende- und Kommunikationspunkte beschrieben werden. Dann gehen wir auf

[1] ***KOM***munikations- und ***PRO***zess***M***onitor für ***I***nformations***S***y***S***teme

die Steuerungsdienste ein, die das erweiterte Steuerprogramm zur Manipulation des Maschinensystems bereitstellt. Schließlich wird auf spezielle Datentypen und Operationen hingewiesen und auf Fragen der Implementierung der Programmiersprache KOMPROMISS eingegangen.

4.1 Sendepunkte

In den Sendepunkten der Algorithmen erfolgt die Kommunikation zwischen den Maschinen des Systems $\mathcal{M}^t$. Diese Kommunikation beinhaltet einerseits den Austausch von Nachrichten und andererseits die Übergabe der Steuerung. Wir beschreiben zunächst den Nachrichtenaustausch und betrachten dann die Steuerungsübergabe.

4.1.1 Nachrichtenaustausch

Bei der Kommunikation zwischen den abstrakten Maschinen unterscheiden wir zwei Formen von Nachrichten:

1. Nachrichten, die die Maschinen des KI-Assistenten untereinander austauschen, und

2. Nachrichten, die die Maschinen des KI-Assistenten mit der Maschine 'IRS' austauschen.

Der Austausch von Nachrichten zwischen den Maschinen des KI-Assistenten ist ein Datenaustausch. Für diesen Datenaustausch stellt KOMPROMISS das Konzept der ***Schnittstelle*** bereit. Eine Schnittstelle ist ein Speicherbereich, dessen Struktur sowohl dem Sender als auch dem Empfänger bekannt ist. Er wird vom Sender mit Daten belegt, die der Empfänger entgegennehmen kann. Aus der Sicht der Programmierpraxis wird die folgende Unterscheidung getroffen:

1. KOMPROMISS stellt eine globale Schnittstelle für alle Maschinen des KI-Assistenten zur Verfügung; jede Maschine hat Zugriff zu dieser Schnittstelle. Sie dient zur Übermittlung von Parametern zwischen Sender und Empfänger und wird deshalb als ***Parameter-Schnittstelle*** bezeichnet.

2. Für den Datenaustausch zwischen den Maschinen eines Teilsystems können in KOMPROMISS ***Teilsystem-Schnittstellen*** eingerichtet, verwendet und wieder gelöscht werden.

Während es nur eine einzige Parameter-Schnittstelle gibt, die beim Aufbau des KI-Assistenten vom erweiterten Steuerprogramm bereitgestellt wird, können die Maschinen des KI-Assistenten mehrere Teilsystem-Schnittstellen einrichten, die jeweils ihre individuelle Lebensdauer haben.

Die Parameter-Schnittstelle wird vom Sender und vom Empfänger in gleicher Weise strukturiert. Das erfolgt in KOMPROMISS durch die folgende Deklarationsanweisung:

```
PARMLAYOUT(<Identifikator>),
   2 ... ,
       3 ... ,
         .
         .
   2 ... ;
```

Der <Identifikator> ist ein PL/I-Bezeichner. Er benennt die erste Ebene der Parameterstruktur und kann zur Präfigierung der Parameterbezeichner verwendet werden. Ab zweiter Ebene wird die Struktur der Parameter-Schnittstelle beschrieben. Soll eine Maschine mit mehreren anderen Maschinen in Kommunikation treten, so kann ihr Programm unterschiedliche Deklarationsanweisungen für die Parameter-Schnittstelle enthalten.

Eine Teilsystem-Schnittstelle dient der problemspezifischen Kommunikation zwischen den Maschinen eines zeitweilig bestehenden Teilsystems. Im Zuge des Aufbaus des Teilsystems wird die Teilsystem-Schnittstelle in einer gewünschten Größe vom erweiterten Steuerprogramm angefordert. Dabei wird ihr ein Kennwort zugeordnet, das sie identifiziert. Die Anweisung, die KOMPROMISS für das Anfordern einer Teilsystem-Schnittstelle bereitstellt, lautet:

```
GETINTERFACE(<Schnittstellenlänge>,<Kennwort>);
```

Die Maschine, die die Schnittstelle eingerichtet hat, kann sie sofort für den Datenaustausch verwenden. Jede andere Maschine erhält erst durch Angabe von <Kennwort> Zugriff auf diese Schnittstelle:

```
USEINTERFACE(<Kennwort>);
```

Die Programme der Maschinen, die über eine Teilsystem-Schnittstelle Daten austauschen wollen, müssen diese Schnittstelle in gleicher Weise strukturieren. Die Strukturbeschreibung der Teilsystem-Schnittstelle erfolgt unter Angabe des Kennwortes in folgender Deklarationsanweisung:

```
DCLINTERFACE (<Kennwort>),
   2 ... ,
       3 ... ,
         .
         .
   2 ... ;
```

Das <Kennwort> benennt wiederum die erste Ebene der Struktur und kann zur Präfigierung der Datenbezeichner verwendet werden.

Ist eine Maschine Kommunikationspartner in mehreren Teilsystemen, so muss sie sich den Zugriff zu den jeweiligen Teilsystem-Schnittstellen verschaffen. Außerdem müssen in das Programm, nach dem die Maschine arbeitet, die Strukturbeschreibungen für alle diese Teilsystem-Schnittstellen eingefügt werden.

Wird das Teilsystem der Maschinen wieder abgebaut, so sollte auch die Teilsystem-Schnittstelle gelöscht werden:

```
RELEASEINTERFACE(<Kennwort>);
```

Für die Kommunikation zwischen den Maschinen des KI-Assistenten einerseits und der Maschine 'IRS' andererseits wird ein spezieller Nachrichtenbereich zur

Verfügung gestellt. Da der Nachrichtenaustausch des Information-Retrieval-Systems STAIRS auf den Benutzer ausgerichtet ist, erfolgt dieser Austausch traditionell in Form von Texten, die im zeilenorientierten Bildschirmformat gegliedert sind. Der Nachrichtenbereich ist deshalb so dimensioniert, dass er eine Bildschirm-Seite mit 24 Zeilen aufnehmen kann.

Der Nachrichtenbereich wird im Zuge des Aufbaus des KI-Assistenten angelegt und steht allen Maschinen als Systemvariable mit der Bezeichnung `MESSAGE` zur Verfügung. Will eine Maschine eine Nachricht in `MESSAGE` hinterlegen, so muss sie diese als Zeichenkette dem Nachrichtenbereich zuweisen. Die Länge der Nachricht wird der Systemvariablen `LMESSAGE` zugewiesen. Die Nachrichten, die die Maschine 'IRS' für die Maschinen des KI-Assistenten bereitstellt, liegen ebenfalls in `MESSAGE` vor und haben die Länge `LMESSAGE`.

Sollen die Kommandos, die die Maschinen des KI-Assistenten an die Maschine 'IRS' absenden wollen, in den Nachrichtenbereich `MESSAGE` eingetragen werden, so sind keine besonderen Hilfsmittel erforderlich, da die Kommandos formatfrei geschrieben werden können. Die Antworten der Maschine 'IRS' sind dagegen formatiert. In jedem Falle weisen sie eine Zeilenstruktur auf. KOMPROMISS ermöglicht den Zugriff auf die i-te Zeile der Nachricht durch die Verwendung des vom System dem Nachrichtenbereich überlagerten Feldes `MESSAGELINE(1:24)`. Eine beliebige andere Struktur des Nachrichtenbereichs kann durch die spezielle Deklaration

```
MESSAGELAYOUT (<Identifikator>),
    2 ... ,
        3 ... ,
          .
          .
    2 ... ;
```

beschrieben werden. Der <Identifikator> benennt dabei wiederum die erste Ebene der Struktur und kann zur Präfigierung der Nachrichtenelemente verwendet werden.

Lässt sich die Struktur des im Nachrichtenbereich hinterlegten Textes nicht durch Formatierung beschreiben, so muss sie durch eine syntaktische Analyse

ermittelt werden. KOMPROMISS unterstützt diesen Prozess durch die Bereitstellung von sprachlichen Mitteln zur Durchführung der lexikalischen Analyse. Diese werden im Abschnitt 4.2.1 beschrieben.

In allen Sendepunkten werden Nachrichten für einen Empfänger bereitgestellt. Dies erfolgt durch Hinterlegen der Nachricht in einer Schnittstelle oder im Nachrichtenbereich. Erst nach der Übergabe der Steuerung an den Empfänger kann dieser auf die Nachrichten zugreifen. In diesem Abschnitt wurden bisher die sprachlichen Mittel beschrieben, die KOMPROMISS für das Hinterlegen der Nachricht bereitstellt. In den folgenden Abschnitten sollen nun die Sprachkonstruktionen zur Steuerungsübergabe vorgestellt werden.

4.1.2 Steuerungsübergabe

4.1.2.1 Generierungspunkt

Wenn eine abstrakte Maschine $m_g = (\mathcal{A}_g, d_g)$ einen Generierungspunkt der Form $s_g = (n, \underline{gen}(\mathcal{A}, I(m))) \in \mathcal{A}_g$ erreicht, so beauftragt sie das erweiterte Steuerprogramm mit der Generierung und gleichzeitigen Aktivierung einer neuen Maschine m, die nach dem Algorithmus $\mathcal{A}$ arbeiten soll. Das erweiterte Steuerprogramm übergibt dabei m_g den Identifikator der neu eingerichteten Maschine. Der Algorithmus $\mathcal{A}$ wird durch ein KOMPROMISS-Programm beschrieben, das durch seinen Programmnamen angegeben wird. Die Maschine m_g muss dem erweiterten Steuerprogramm mitteilen, wohin dieses den Identifikator $I(m)$ speichern soll. In den meisten Fällen wird sie ihn in das Datenobjekt d_g speichern wollen, sie kann ihn aber auch in eine Schnittstelle eintragen lassen. Als Ergänzung der von PL/I bereitgestellten Datentypen führt KOMPROMISS zusätzlich den neuen Datentyp `PROCESS` ein, der zur Aufnahme des Identifikators einer Maschine dient.

Die Steuerungsübergabe im Generierungspunkt wird wie folgt angewiesen:

`GENERATE` (<Programmname>`,`<`PROCESS`-Variable>) `;`

Das erweiterte Steuerprogramm weist den Identifikator der neu eingerichteten Maschine m der PROCESS-Variablen zu.

Eine vereinfachte Variante des Generierungspunktes stellt der Unterprogramm-Aufrufpunkt $s_{ua} = (n, \underline{exec}(\mathcal{A})) \in \mathcal{A}_g$ dar, in dem eine Maschine generiert und aktiviert wird, deren Algorithmus $\mathcal{A}$ vom Typ einer Routine ist. Diese Steuerungsübergabe wird in KOMPROMISS durch den Befehl

```
EXECUTE(<Programmname>);
```

angewiesen. Der Befehl bewirkt, dass das angegebene Programm als Routine ausgeführt wird.

4.1.2.2 Aktivierungspunkt

In einem Aktivierungspunkt $s_a = (n, \underline{act}(I(m)))$ wird eine schon existierende Maschine m zur Fortsetzung ihrer Arbeit aufgefordert. Die Übergabe der Steuerung wird durch

```
ACTIVATE(<PROCESS-Variable>);
```

angewiesen. Dabei muss die PROCESS-Variable den Identifikator $I(m)$ als Wert besitzen.

Handelt es sich bei m um die Maschine 'IRS', dann wurde sie vom erweiterten Steuerprogramm eingerichtet, das ihren Identifikator $I('IRS')$ der global verfügbaren PROCESS-Variablen mit der Bezeichnung IRS zugewiesen hat. Die Anweisung

```
ACTIVATE(IRS);
```

bewirkt somit die Übergabe der Steuerung an die Maschine 'IRS', also die Aktivierung des Information-Retrieval-Systems STAIRS.

4.1.2.3 Aktivator-Rückkehrpunkt

Für einen Aktivator-Rückkehrpunkt $s_{ar} = (n, \underline{aret})$ stellt KOMPROMISS das folgende Kommando zur Verfügung:

```
READY;
```

Das erweiterte Steuerprogramm gibt dann die Steuerung an den aktuellen Aktivator zurück.

4.1.2.4 Endpunkt

Erreicht der Algorithmus einer abstrakten Maschine $m = (\mathcal{A}, d)$ seinen Endpunkt $s^{+}_{\mathcal{A}} = (n, \underline{gret^{+}})$, so beendet die Maschine m ihre Existenz und die Steuerung wird an ihren Generator zurückgegeben. In einem KOMPROMISS-Programm wird das durch die letzte Anweisung ausgedrückt, die in einer der beiden Formen

```
%INCLUDE PROCEND;
```

oder

```
%INCLUDE ROUTEND;
```

angegeben werden muss. Die erste Form wird dabei bei Algorithmen vom Typ eines Prozesses und die zweite bei Algorithmen vom Typ einer Routine verwendet.

Im Modell der abstrakten Maschinen sind zwei weitere Fälle der Steuerungsübergabe im Endpunkt beschrieben worden: die Steuerungs-Rückgabe an den Aktivator und die Übergabe an eine beliebige andere Maschine. Für diese Fälle

stellt KOMPROMISS aus den im Abschnitt 3.3 erläuterten Gründen keine speziellen Anweisungen zur Verfügung. Durch die Kombination eines Aktivator-Rückkehrpunktes bzw. eines Aktivierungspunktes mit einer Anweisung zum „gewaltsamen“ Löschen der Maschine m lassen sich diese Fälle bei Bedarf in einfacher Weise programmieren.

4.2 Kommunikationspunkte

Jede Nachricht, die der KI-Assistent mit dem Benutzer austauscht, wird im Nachrichtenbereich `MESSAGE` hinterlegt bzw. diesem entnommen. Sie enthält im allgemeinen verschiedene Elemente. Diese Elemente stehen entweder an fest vorgegebenen Positionen oder sind durch ihre spezielle Einbettung in den Kontext einer syntaktischen Konstruktion bestimmt.

Der Zugriff zu jenen Nachrichtenelementen, die sich an festen Positionen im Nachrichtenbereich `MESSAGE` befinden, wird durch eine Strukturbeschreibung von `MESSAGE` unterstützt. Die dafür vorgesehene Deklarationsanweisung `MESSAGELAYOUT` wurde bereits im Abschnitt 4.1.1 angegeben.

Im Eintrittspunkt und in den Dialogpunkten sind jedoch unformatierte Eingaben des Benutzers auszuwerten. Für die Analyse dieser Eingaben stellt KOMPROMISS zwei Techniken bereit: die lexikalische Analyse und die Masken-Technik. Diese Techniken werden in den folgenden Abschnitten beschrieben.

4.2.1 Lexikalische Analyse

KOMPROMISS bietet sprachliche Werkzeuge an, mit deren Hilfe unformatierte Nachrichten in ihre Strukturelemente zerlegt werden können. Der Vorgang der Zerlegung wird als lexikalische Analyse bezeichnet, die ermittelten lexikalischen Einheiten werden Morpheme genannt. Die lexikalische Analyse kann erfolgen, um die von der Maschine 'IRS' gesendeten unformatierten Nachrichten zu analysieren oder um die – stets unformatierten - Benutzereingaben auszuwerten.

Zur Beschreibung der lexikalischen Analyse verwenden wir die folgenden Begriffe und Symbole aus der Theorie der formalen Sprachen:[1]

1. Ein Alphabet ist eine endliche nichtleere Menge von Symbolen.

2. Ist Z ein Alphabet, so nennen wir eine Folge $w = z_1 z_2 ... z_n$ von Symbolen $z_i \in Z;\ i = 1,2,\ldots,n$ ein Wort über Z.

3. Die Zusammensetzung des Wortes $x = x_1 x_2 ... x_n;\ x_i \in Z;\ i = 1,2,\ldots,n$ mit dem Wort $y = y_1 y_2 ... y_m;\ y_i \in Z;\ i = 1,2,\ldots,m$ ist definiert durch den Ausdruck $xy = x_1 x_2 \ldots x_n y_1 y_2 \ldots y_m$.

4. Bezüglich der Operation „Zusammensetzung“ wird als neutrales Element das Leerwort eingeführt und mit ε bezeichnet: $\varepsilon w = w\varepsilon = w$.

5. Sind u, v, w und x Wörter mit $x=uvw$, so heißt v Teilwort von x, im Falle $u = \varepsilon$ auch Präfix von x und im Falle $w = \varepsilon$ auch Suffix von x.

6. Das Produkt zweier Alphabete X und Y ist festgelegt durch

$$XY = \{ xy \mid x \in X \wedge y \in Y \}$$

7. Die Potenz eines Alphabets Z wird erklärt durch

$$Z^0 = \{\varepsilon\}, \quad Z^1 = Z \quad \text{und} \quad Z^n = Z^{n-1} Z \ \text{für}\ n > 1$$

8. Die Sternhülle und die Plushülle eines Alphabets Z werden definiert durch

[1] Vgl. beispielsweise Mayer (1978).

$$Z^* = \bigcup_{n \geq 0} Z^n \qquad Z^+ = \bigcup_{n \geq 1} Z^n$$

Der durch die lexikalische Analyse auszuwertende Text ist ein Wort $w = z_1 z_2 \ldots z_n$ über dem Alphabet Z des ASCII-Zeichensatzes. Das Alphabet Z besteht aus dem Leerzeichen "$\circ$", dem Apostroph, dem Teilalphabet L der Buchstaben, dem Teilalphabet D der Ziffern, dem Teilalphabet S ausgewählter Sonderzeichen und dem Teilalphabet S_1 der restlichen Zeichen des ASCII-Zeichensatzes:

$$\begin{aligned} Z &= \{\circ\} \cup \{'\} \cup L \cup D \cup S \cup S_1 \\ L &= \{A, B, \ldots, Z, a, b, \ldots, z\} \\ D &= \{0,1,2,3,4,5,6,7,8,9\} \\ S &= \{+, -, *, /, ., ;, (,), =, ?\} \cup \{,\} \end{aligned} \tag{29}$$

Durch die Werkzeuge der lexikalischen Analyse werden aus dem Text w Teilwörter extrahiert - die Morpheme. Auf Grund ihrer Struktur werden die Morpheme vordefinierten Morphemklassen zugeordnet. KOMPROMISS unterscheidet die folgenden Morphemklassen:

Bezeichner:	$QTAG = L\,(L \cup D)^*$
Natürliche Zahl:	$QNUMBER = D^+$
String:	$QSTRING = \{'\}\ (Z - \{'\})^*\ \{'\}$
Pluszeichen:	$QPLUS = \{+\}$
Minuszeichen:	$QMINUS = \{-\}$
Multiplikationszeichen:	$QTIMES = \{*\}$
Divisionszeichen:	$QDIVISION = \{/\}$
Punkt:	$QPOINT = \{.\}$

Komma:	$QCOMMA = \{ , \}$
Semikolon:	$QSEMICOLON = \{ ; \}$
Öffnende Klammer:	$QOPEN = \{ (\}$
Schließende Klammer:	$QCLOSE = \{) \}$
Gleichheitszeichen:	$QEQUAL = \{ = \}$
Fragezeichen:	$QQUESTION = \{ ? \}$
Sonstiges Sonderzeichen:	$QNOTDEF = S_1$
Nachrichtenende:	QEND

Der Morphemklasse QEND entspricht keine Morphemstruktur. Sie wird nur eingeführt, um das Ende des Textes zu repräsentieren.

KOMPROMISS geht von dem Konzept aus, dass sich ***Leseköpfe*** entlang eines zu analysierenden Textes bewegen. Jeder Lesekopf ist mit einem individuellen Analyse-Prozess verbunden. Die Leseköpfe bewegen sich in diskreten Zeitpunkten vorwärts und ermitteln das jeweils nächste Morphem des zu analysierenden Textes. Die Bewegung verschiedener Leseköpfe erfolgt unabhängig voneinander: Sie können in ihrem Analyse-Stand verharren, zur Fortsetzung der Analyse „angestoßen" werden und sich gegenseitig „überholen". Leseköpfe können in einem erreichten Analyse-Stand auch kopiert werden, so dass mehrere Exemplare des Lesekopfes entstehen, die unabhängig voneinander die lexikalische Analyse nach verschiedenen Gesichtspunkten fortsetzen können. Durch diese Arbeitsweise lässt sich bei der syntaktischen Analyse eines Textes in einfacher Weise ein „Backtracking"[1] realisieren.

Ein symbolischer Lesekopf ist ein Quadrupel

$$\lambda = \left(y, \mu, \kappa_\mu, \eta \right) \tag{30}$$

[1] Vgl. beispielsweise Russell/Norvig (2004), S. 280 ff.

Dabei bedeuten:

y	- den zu analysierenden Rest des Textes,
μ	- das zuletzt gelesene Morphem des Textes,
κ_μ	- die Morphemklasse, zu der das Morphem μ gehört, und
η	- eine Markierung.

Die Notwendigkeit der Markierung η ergibt sich aus folgender Überlegung: Die lexikalische Analyse geht von der Voraussetzung aus, dass die Verarbeitung eines Morphems in Abhängigkeit von der Morphemklasse erfolgt, der es angehört. Deshalb ermöglicht KOMPROMISS die Abfrage, ob das aktuell gelesene Morphem einer vermuteten Morphemklasse angehört. Ist das der Fall, wird das Morphem als „erkannt" markiert (Schreibweise: $\eta := \mathbf{true};$), ansonsten als „nicht erkannt" (Schreibweise: $\eta := \mathbf{false};$).

Zur Durchführung der lexikalischen Analyse eines Textes w wird ein Lesekopf initialisiert. Er hat zum Zeitpunkt t_0 die Form: $\lambda^{t_0} = (w, \otimes, \otimes, \mathbf{true})$. Der gesamte Text ist (noch) zu analysieren. Es wurde noch kein Morphem gelesen; deshalb sind das Morphem μ sowie die Morphemklasse κ_μ unbestimmt, was durch das Symbol "$\otimes$" ausgedrückt wird. Durch die Initialmarkierung **true** wird erreicht, dass der Lesekopf λ^{t_0} bei seiner ersten Aktivierung sofort das erste Morphem liest.

Im folgenden beschreiben wir die Algorithmen, nach denen in KOMPROMISS die lexikalische Analyse des Textes w mit Hilfe des Lesekopfes λ erfolgt. Dabei betrachten wir den Zeitpunkt t, in dem vom Algorithmus einer Maschine geprüft wird, ob das zuletzt gelesene Morphem der Morphemklasse κ^*_μ angehört.

Zur Beschreibung der Algorithmen verwenden wir eine formale Schreibweise, die an PASCAL angelehnt ist:[1]

[1] Auf das Schlüsselwort **Algorithmus:** folgt der Name des Algorithmus. Daran schließen sich die Eingabeparameter (nach dem Schlüsselwort **Eingabe:**) und die Ausgabeparameter (nach dem Schlüsselwort **Ausgabe:**) an. Auf das Schlüsselwort **Vorgehen:** folgt die Beschreibung des

Algorithmus: Morphemklasse prüfen

Eingabe: Lesekopf $\lambda^{t-1} = \left(y^{t-1}, \mu^{t-1}, \kappa_{\mu}^{t-1}, \eta^{t-1}\right)$

zu prüfende Morphemklasse κ_{μ}^{*}

Ausgabe: Lesekopf $\lambda^{t} = \left(y^{t}, \mu^{t}, \kappa_{\mu}^{t}, \eta^{t}\right)$

Funktionswert $\text{result} := \textbf{true}$, falls das Morphem μ^{t} zur Klasse κ_{μ}^{*} gehört, $\text{result} := \textbf{false}$ sonst

Vorgehen:

co Wurde das letzte Morphem bereits richtig „erkannt“, dann wird das nächste Morphem eingelesen. Die aktuelle Morphemklasse wird mit der zu prüfenden Morphemklasse verglichen.
oc

begin
if η^{t-1}
then $\text{Morphem extrahieren}\left(y^{t-1}, \left(y^{t}, \mu^{t}, \kappa_{\mu}^{t}\right)\right)$
else
begin $y^{t} := y^{t-1}$; $\mu^{t} := \mu^{t-1}$; $\kappa_{\mu}^{t} := \kappa_{\mu}^{t-1}$; **end**;
$\eta^{t} := \left(\kappa_{\mu}^{t} = \kappa_{\mu}^{*}\right)$; $\text{result} := \eta^{t}$;
end

Im Algorithmus 'Morphemklasse prüfen' erfolgt die Extraktion des jeweils nächsten Morphems immer dann, wenn die Klassenzugehörigkeit des zuletzt

---

Informationsverarbeitungsprozesses in einer PASCAL-ähnlichen Sprache. Für die im Algorithmus verwendeten Datentypen sind keine Beschränkungen vorgegeben. Als Bezeichner der Variablen dienen die in der theoretischen Beschreibung verwendeten Symbole. Lokale Variable werden nach dem Schlüsselwort **var** aufgeführt. Die Strukturierung der Algorithmen entspricht der üblichen PASCAL-Notation. Für die Anweisungen können auch fachsprachliche Formulierungen verwendet werden. Kommentare werden durch Klammerung mit Hilfe der Schlüsselwörter **co** und **oc** kenntlich gemacht.

gelesenen Morphems bereits „erkannt“ wurde. Dieser standardmäßige Verlauf der lexikalischen Analyse lässt sich in zweierlei Weise abändern:

1. Obwohl das zuletzt gelesene Morphem μ^{t-1} noch „nicht erkannt“ wurde, wird es dennoch als „erkannt“ markiert. Dadurch wird das Morphem μ^{t-1} freigegeben und bei der nächsten Aktivierung des Lesekopfes das nächste Morphem gelesen.

2. Obwohl das zuletzt gelesene Morphem μ^{t-1} schon „erkannt“ wurde, wird es dennoch als „nicht erkannt“ markiert. Damit besteht die Möglichkeit, seine Klassenzugehörigkeit noch einmal zu prüfen. Das führt oft zu einfacheren Algorithmen bei der syntaktischen Analyse.[1]

Der Algorithmus 'Morphemklasse prüfen' verwendet den Algorithmus 'Morphem extrahieren', der nun angegeben wird. Dabei steht das Symbol "$\circ$" für das Leerzeichen.

Algorithmus: Morphem extrahieren

Eingabe: Rest der Nachricht y^{t-1}

Ausgabe: Rest der Nachricht y^{t}
gelesenes Morphem μ^{t}
Morphemklasse des gelesenen Morphems κ_{μ}^{t}

Vorgehen:

co Führende Leerzeichen werden überlesen. Entsprechend der Präfixstruktur werden der Rest der Nachricht, das nächste Morphem und seine Morphemklasse ermittelt.
oc

[1] Insbesondere dann, wenn ein gegebenes Morphem einerseits eine syntaktische Gruppe beendet und andererseits die folgende syntaktische Gruppe einleitet.

var y;

begin

$y := y^{t-1}$; **while** $y = \circ y'$ **do** $y := y'$;

if $\left(y = zwy' \text{ mit } z \in L, w \in (L \cup D)^*\right)$ **then**

begin $y^t := y'$; $\mu^t := zw$; $\kappa_\mu^t := QTAG$; **end else**

if $\left(y = wy' \text{ mit } w \in D^+\right)$ **then**

begin $y^t := y'$; $\mu^t := w$; $\kappa_\mu^t := QNUMBER$; **end else**

if $\left(y = 'w' y' \text{ mit } w \in (Z - \{'\})^*\right)$ **then**

begin $y^t := y'$; $\mu^t := w$; $\kappa_\mu^t := QSTRING$; **end else**

if $(y = +y')$ **then**

begin $y^t := y'$; $\mu^t := +$; $\kappa_\mu^t := QPLUS$; **end else**

if $(y = -y')$ **then**

begin $y^t := y'$; $\mu^t := -$; $\kappa_\mu^t := QMINUS$; **end else**

. . .

if $(y = ?y')$ **then**

begin $y^t := y'$; $\mu^t := ?$; $\kappa_\mu^t := QQUESTION$; **end else**

if $(y = zy' \text{ mit } z \in S_1)$ **then**

begin $y^t := y'$; $\mu^t := z$; $\kappa_\mu^t := QNOTDEF$; **end else**

if $(y = \varepsilon)$ **then**

begin $y^t := \varepsilon$; $\mu^t := \otimes$; $\kappa_\mu^t := QEND$; **end**;

end

Nachdem das methodische Konzept der lexikalischen Analyse erläutert wurde, stellen wir nun die Sprachkonstruktionen vor, die KOMPROMISS dafür bereitstellt.

Die lexikalische Analyse kann auf Texte angewendet werden, die in drei Speicherformen vorliegen:

1. als Nachricht im Nachrichtenbereich MESSAGE,

2. als Text in einer Variablen mit dem PL/I-Datenattribut CHARACTER und

3. als Text in einer Variablen mit dem KOMPROMISS-Datenattribut HEAPSTRING (dieses kennzeichnet einen String, der auf einer globalen Halde abgelegt ist - s. Abschnitt 4.4.2).

Eine Variable, die einen Lesekopf aufnehmen soll, wird mit dem KOMPROMISS-Datenattribut READER deklariert. Vom System wird ein Lesekopf mit der Bezeichnung READERMESSAGE bereits vordefiniert. Er steht allen Maschinen des KI-Assistenten zur Verfügung und kann für die lexikalische Analyse des Nachrichtenbereichs MESSAGE verwendet werden.

Ein Lesekopf, der für eine lexikalische Analyse verwendet werden soll, muss initialisiert werden. Der Lesekopf READERMESSAGE wird vom System automatisch initialisiert, wenn die Maschine 'IRS' oder der Benutzer eine Nachricht in MESSAGE hinterlegt. Soll die Analyse von MESSAGE jedoch noch einmal von vorn erfolgen, so muss eine erneute Initialisierung von READERMESSAGE explizit vorgenommen werden. Für die Initialisierung eines Lesekopfes stellt KOMPROMISS in Abhängigkeit von den genannten drei Speicherformen der zu analysierenden Texte die folgenden Anweisungen bereit:

1. SCANMESSAGE;

2. SCANCHARSTRING(<CHARACTER-Variable>,
   <Textlänge>,<READER-Variable>);

3. SCANHEAPSTRING(<HEAPSTRING-Variable>,
   <READER-Variable>);

Die Prüfung, ob das zuletzt gelesene Morphem einer bestimmten Morphemklasse angehört, erfolgt mit Hilfe der logischen Funktionen

1. `MORPHEM(`<Morphemklasse>`)`

2., 3. `READERMORPHEM(`<READER-Variable>`,`
<Morphemklasse>`)`

Liefert die Funktion den Wert **true**, so steht das Morphem in der Systemvariablen `INSTRING` mit der Länge `LINSTRING` bereit.

Durch die KOMPROMISS-Anweisungen

1. `RELEASEMORPHEM;`

2., 3. `RELEASEREADERMORPHEM(`<READER-Variable>`);`

wird ein noch „nicht erkanntes“ Morphem freigegeben. Das Festhalten eines bereits „erkannten“ Morphems erfolgt mit Hilfe der Anweisungen

1. `HOLDMORPHEM;`

2., 3. `HOLDREADERMORPHEM(`<READER-Variable>`);`

4.2.2 Eintrittspunkt

Will der Benutzer in seine Kommunikation mit dem Information-Retrieval-System einen KI-Assistenten einbeziehen, so wendet er sich mit dem Kommando

##<Programmname der Monitormaschine> <Text>

an das erweiterte Steuerprogramm, das daraufhin eine Monitormaschine zum angegebenen Algorithmus generiert und aktiviert. Am Beginn dieses Algo-

rithmus liegt der Eintrittspunkt des KI-Assistenten. Dort erhält die Monitormaschine im Nachrichtenbereich `MESSAGE` die Information mit der Länge `LMESSAGE`, die der Benutzer als <Text> eingegeben hat.

4.2.3 Dialogpunkt

In einem Dialogpunkt sendet eine Maschine des KI-Assistenten eine Nachricht an den Benutzer und empfängt gegebenenfalls dessen Antwort. Sowohl die Nachricht als auch die Antwort stehen dabei im Nachrichtenbereich `MESSAGE` und haben jeweils die in `LMESSAGE` gespeicherte Länge. Die Nachricht an den Benutzer ist dazu bestimmt, auf dem Bildschirm ausgegeben zu werden. Als Angabe, wo die Nachricht auf dem zeilenorientierten Bildschirm beginnen soll, werden in KOMPROMISS die Schlüsselwörter `NEXTPOS` (an der nächsten Position in der aktuellen Zeile), `NEWLINE` (am Beginn der folgenden Zeile) bzw. `NEWPAGE` (am Beginn einer neuen Bildschirm-Seite) verwendet.

Soll lediglich eine Nachricht ausgegeben werden, ohne dass eine Antwort des Benutzers erwartet wird, so lautet die KOMPROMISS-Anweisung:

```
TELLUSER( { NEXTPOS } );
          { NEWLINE }
          { NEWPAGE }
```

Soll der Benutzer dem KI-Assistenten nach Erhalt der Nachricht eine Antwort zuschicken, so wird das in KOMPROMISS wie folgt angewiesen:

```
ASKUSER( { NEXTPOS } );
         { NEWLINE }
         { NEWPAGE }
```

Soll mit der Ausgabe der Nachricht auch ein akustisches Signal gesendet werden, so steht dafür die folgende Anweisung zur Verfügung:

```
WAKEANDASKUSER( { NEXTPOS } );
               { NEWLINE }
               { NEWPAGE }
```

Für die Analyse der Antworten des Benutzers stellt KOMPROMISS zwei Techniken bereit: die im Abschnitt 4.2.1 beschriebene lexikalische Analyse und die Masken-Technik (Formular-Prinzip). Bei der Masken-Technik besteht die Nachricht, die dem Benutzer geschickt wird, einerseits aus der Aufforderung, ein bestimmtes Formular auszufüllen, und andererseits aus dem Formular selbst. Die Bildschirmpositionen, an denen die gefragten Daten einzugeben sind, werden auf dem Bildschirm besonders ausgewiesen. Das kann entweder durch vorgeschlagene Antworten (Offerten) im geforderten Datenformat oder durch Platzhalter (beispielsweise Punkte) erfolgen. Der Benutzer kann die Offerten akzeptieren (unverändert lassen) oder durch Überschreiben der Offerten bzw. Platzhalter seine Daten eintragen. Aufforderung und Formular werden unter Verwendung einer Strukturdeklaration der Form

```
MESSAGELAYOUT (<Identifikator>),
   2 AUFFORDERUNG,
        3 ... ,
          .
          .
   2 FORMULAR,
        3 ... ,
          .
          .
        3 ... ;
```

in den Nachrichtenbereich `MESSAGE` eingetragen. Die Länge der Aufforderung wird `LMESSAGE` und die Länge des Formulars der Systemvariablen `LOFFER` zugewiesen.

Bei Verwendung der Masken-Technik hat der Dialogpunkt in KOMPROMISS die Form

```
ASKANDOFFERUSER( { NEXTPOS | NEWLINE | NEWPAGE } );
```

Nach der Ausführung dieses Kommandos können die Daten des Formulars den Variablen der Teilstruktur `FORMULAR` entnommen werden.

Die Technik, dem Benutzer eine Offerte zuzuschicken, kann auch für nichtformatierte Informationen - also beispielsweise für Texte - verwendet werden. Das Formular besteht dann lediglich aus einem Textfeld. Der Benutzer erhält dieses Textfeld auf dem Bildschirm angezeigt und kann es verändern. Die Auswertung des - eventuell veränderten - Textfeldes erfolgt dann mit den Mitteln der lexikalischen Analyse. Der Lesekopf `READERMESSAGE` wird dabei vom System automatisch auf den Beginn dieses Textfeldes positioniert.

4.2.4 Austrittspunkt

Erreicht eine Maschine des KI-Assistenten einen Austrittspunkt, so informiert sie den Benutzer darüber, dass der KI-Assistent aufhört zu existieren und dass der Benutzer wieder den direkten Dialog mit dem Information-Retrieval-System aufnehmen soll. Diese Nachricht des KI-Assistenten - gewissermaßen sein „Testament“ - wird im Nachrichtenbereich `MESSAGE` hinterlegt und dem Benutzer durch folgende KOMPROMISS-Anweisung zugeschickt:

```
LASTWILLFORUSER( { NEXTPOS | NEWLINE | NEWPAGE } );
```

4.3 Steuerungsdienste

Im Abschnitt 3.4.5 wurde bereits darauf hingewiesen, dass es für eine Maschine des KI-Assistenten die Möglichkeit geben muss, das erweiterte Steuerprogramm zu befragen, ob eine bestimmte Maschine noch Bestandteil des KI-Assistenten ist. KOMPROMISS stellt dafür die logische Funktion

```
ISWAITING(<PROCESS-Variable>)
```

zur Verfügung. Sie liefert den Wert **true**, wenn die Maschine, deren Identifikator durch <PROCESS-Variable> angegeben wird, auf eine nächste Aktivierung wartet. Ansonsten liefert sie den Wert **false**.

Erweist es sich als notwendig, eine bestimmte Maschine „gewaltsam“ aus dem KI-Assistenten zu entfernen, so kann dafür die folgende KOMPROMISS-Anweisung verwendet werden:

```
TERMINATE(<PROCESS-Variable>);
```

4.4 Spezielle Datentypen

Im Zuge der Programmierung von Algorithmen, nach denen die Maschinen des KI-Assistenten arbeiten sollen, ist häufig die Nutzung von Magazinspeichern und Listen sowie die Verwaltung von Strings - also von Zeichenketten variabler Länge auf einer globalen Halde (Heap[1]) - erforderlich. Deshalb erweitert die Programmiersprache KOMPROMISS die Wirtssprache PL/I dahingehend, dass diese Aufgaben effektiv gelöst werden können. Diese Sprachkonzepte sollen nun vorgestellt und an Hand von Beispielprogrammen veranschaulicht werden.

4.4.1 Listen

In einer Liste wird eine zeitlich variierende Anzahl von Elementen zusammengefasst, deren aktuelle Folge sich aus den Einspeicherungen und Streichungen ergibt, die im Zeitverlauf durchgeführt wurden.

In eine Liste können beliebige Werte mit einer `BIN-FIXED(15)`-Struktur als Elemente aufgenommen werden: Neben den Variablen, die explizit mit dem PL/I-Attribut `BIN FIXED(15)` deklariert wurden, sind das zusätzlich auch die `PROCESS`-Variablen, die `HEAPLIST`-Variablen (s. in diesem Abschnitt)

[1] Vgl. beispielsweise Ernst (2003), S. 519 ff.

sowie die HEAPSTRING-Variablen (s. Abschnitt 4.4.2). Da als ein Element einer Liste wiederum eine Liste fungieren kann, sind beliebig verschachtelte Listenstrukturen darstellbar.

Für die Speicherung von Listen wird ein Speicherbereich (Listenheap) benötigt, der durch die Anweisung

```
GETLISTHEAP(<Listenanzahl>,<mittlere-Länge>);
```

eingerichtet werden muss. Dabei ist <Listenanzahl> die geschätzte Anzahl der benötigten Listen und <mittlere-Länge> die mittlere Anzahl ihrer Elemente. Das Produkt der beiden Größen ergibt die globale Maximalanzahl von Listenelementen.

Eine Liste wird mit dem Datentyp HEAPLIST vereinbart:

```
DCL <HEAPLIST-Variable> [<Dimension>] HEAPLIST;
```

Vor der ersten Verwendung einer Liste muss sie initialisiert werden - sie wird dann zur leeren Liste:

```
INITLIST(<HEAPLIST-Variable>);
```

Ob eine Liste leer ist, kann durch die folgende logische Funktion abgefragt werden:

```
IF ISINITLIST(<HEAPLIST-Variable>) THEN ...;
```

Soll ein neuer Wert <Value> mit einer BIN-FIXED(15)-Struktur an das Ende einer Liste angehängt werden, so erfolgt das durch die Anweisung:

```
PUSHLIST(<HEAPLIST-Variable>,<Value>);
```

Wird dabei die durch `GETLISTHEAP` angeforderte Maximalzahl von Listenelementen überschritten, so führt das zu einem abnormalen Ende der Aufgabenbearbeitung.

Der in eine Liste zuletzt eingespeicherte Wert wird durch die Anweisung

```
POPLIST(<HEAPLIST-Variable>,<Var>);
```

der Variablen <Var> zugewiesen und zugleich aus der Liste gestrichen. Wird `POPLIST` auf eine leere Liste angewendet, so wird der Systemzustand `EOF` auf **true** gesetzt.

Die beiden Kommandos `PUSHLIST` und `POPLIST` können vorteilhaft verwendet werden, um einen Magazinspeicher zu realisieren (LIFO-Speicher[1]).

Die aktuelle Länge einer Liste (Anzahl ihrer Elemente) wird durch

```
LENGTHLIST(<HEAPLIST-Variable>,<Länge>);
```

der Variablen <Länge> zugewiesen.

Für das konsekutive Bereitstellen sämtlicher Elemente einer Liste kann ein Listen-Browse-Prozess wie folgt eingerichtet werden:

```
PARMBROWSELIST(<HEAPLIST-Variable>,<Var>);
GENERATE(*BRLIST*,<PROCESS-Variable>);
```

[1] LIFO – ***L***ast ***I***n ***F***irst ***O***ut.

Dabei muss <Var> eine Variable vom Typ BIN-FlXED(15) sein. Ihr wird beim Generieren des Prozesses der Wert des letzten Listenelements zugewiesen. Ist die Liste leer, wird stattdessen der Systemzustand EOF auf **true** gesetzt. Bei jeder Aktivierung des Prozesses durch die Anweisung

```
AVTIVATE(<PROCESS-Variable>);
```

wird <Var> mit dem Wert des jeweils vorangehenden Listenelements belegt. Der Prozess endet, wenn alle Elemente der Liste bereitgestellt wurden. Beim Versuch, ein weiteres Element bereitzustellen, wird der Systemzustand EOF auf **true** gesetzt. Im Listen-Browse-Prozess wird am Anfang das letzte Element und am Ende das erste Element der Liste bereitgestellt (Browse von „hinten" nach „vorn"). Im Gegensatz zum Bereitstellen der Listenelemente durch das Kommando POPLIST, bei dem das ausgelesene Element aus der Liste gestrichen wird, bleibt die Liste beim Listen-Browse-Prozess unverändert.

Ein aktiver Listen-Browse-Prozess kann zwangsweise beendet werden, ehe alle Elemente der Liste bereitgestellt wurden:

```
TERMINATE(<PROCESS-Variable>);
```

Es besteht die Möglichkeit, parallel für mehrere Listen je einen Listen-Browse-Prozess durchzuführen, wobei das Ansteuern der einzelnen Listen-Browse-Prozesse durch die jeweilige <PROCESS-Variable> erfolgt.

Die Reihenfolge der Elemente in einer Liste kann durch die Anweisung

```
REVERSELIST(<HEAPLIST-Variable>);
```

umgedreht werden. Das ist beispielsweise erforderlich vor dem Starten eines Listen-Browse-Prozesses, wenn die Elemente der Liste in der Reihenfolge ihrer Einspeicherung bereitgestellt werden sollen.

Eine nicht mehr benötigte Liste sollte wieder freigesetzt werden, wobei die Anweisung

```
RELEASELIST(<HEAPLIST-Variable>);
```

das Kommando `INITLIST(<HEAPLIST-Variable>)` impliziert.

Ist keine weitere Arbeit mit Listen (auch kein Listen-Browse-Prozess) mehr erforderlich, sollte der Listenheap durch die Anweisung

```
RELEASELISTHEAP;
```

wieder freigegeben werden.

Zur Demonstration der Listenverarbeitung wollen wir einen KI-Assistenten für einen Fachinformator (Information Professional[1]) einrichten. Das geschieht, indem der Fachinformator während seines Dialogs mit dem Information-Retrieval-System STAIRS das folgende Kommando eingibt:

```
##USERLISTS
```

Das erweiterte Steuerprogramm, das die Kommandos des Fachinformators bisher an das Information-Retrieval-System STAIRS „durchgereicht“ hat, erkennt auf Grund der speziellen Syntax des Kommandos, dass nun ein KI-Assistent eingerichtet werden soll. Es generiert und aktiviert eine Monitormaschine, die nach dem Algorithmus `USERLISTS` arbeitet:

[1] Vgl. beispielsweise Lewandowski (2005), S. 36.

```
1  /*********************************************/
2  USERLISTS: %INCLUDE ROUTINE;
3  /*********************************************/
4
5   GETLISTHEAP(3,100);
6   EXECUTE('HANDLELISTS');
7   RELEASELISTHEAP;
8
9   MESSAGELINE(1)=
10  'Jetzt arbeiten Sie wieder mit STAIRS.';
11  MESSAGELINE(2)=
12  'Geben Sie ein STAIRS-Kommando ein!';
13  LMESSAGE=160; LASTWILLFORUSER(NEWPAGE);
14
15 %INCLUDE ROUTEND;
```

Anmerkungen zu den Programmzeilen:

2 Der Algorithmus `USERLISTS` wird als Routine vereinbart.

5 Ein Listenheap für 3 Listen mit im Mittel 100 Elementen wird angefordert.

6 Eine neue Maschine wird generiert und aktiviert, die nach dem Algorithmus der Routine `HANDLELISTS` arbeiten soll.

7 Der Listenheap wird wieder freigegeben.

9-13 Das „Testament“ der Monitormaschine mit der Länge `LMESSAGE` wird in zwei Zeilen des Nachrichtenbereichs `MESSAGE` hinterlegt und auf einer neuen Bildschirm-Seite ausgegeben. Der KI-Assistent wird daraufhin abgebaut, und der Benutzer tritt wieder in die direkte Kommunikation mit dem Information-Retrieval-System STAIRS.

Die Routine `HANDLELISTS` erstellt zunächst aus den Angaben des Fachinformators eine Liste von Dokumenten-Nummern (`TOTALLIST`), wie sie im Ergebnis einer Recherche entstehen könnte. Dann werden die Dokumenten-

Nummern dem Fachinformator nacheinander angezeigt, damit er sie in zwei Dokumentenlisten für die Benutzer „Anton“ (ANTONLIST) und „Berta“ (BERTALIST) einordnen kann. Schließlich werden die beiden Dokumentenlisten für „Anton“ und „Berta“ nacheinander ausgegeben.

```
/*********************************************/
HANDLELISTS: %INCLUDE ROUTINE;
/*********************************************/

 DCL (TOTALLIST,ANTONLIST,BERTALIST) HEAPLIST,
     ASKNEXTDOC BIT(1);

 DCL SHOWLIST ENTRY(CHAR(*),HEAPLIST);
 SHOWLIST: PROCEDURE(USER,USERLIST);
  DCL  USER CHAR(*),
       USERLIST HEAPLIST;
  DCL  BROWSE PROCESS,
       NDOCUMENTS PIC '999',
       DOC_NR PIC '999';

  LENGTHLIST(USERLIST,NDOCUMENTS);
  MESSAGELINE(1)=NDOCUMENTS||
                 ' Dokumente für Benutzer '||
                 USER||':';
  MESSAGELINE(2)=' '; LMESSAGE=160;
  TELLUSER(NEWPAGE);

  REVERSELIST(USERLIST);
  PARMBROWSELIST(USERLIST,DOC_NR);
  GENERATE(*BRLIST*,BROWSE);
  DO WHILE(¬EOF);
     SUBSTR(MESSAGE,1,20)='Dokument='||DOC_NR;
     LMESSAGE=20; TELLUSER(NEXTPOS);
     ACTIVATE(BROWSE);
  END;
```

```
31
32   MESSAGELINE(1)=' ';
33   MESSAGELINE(2)='Ende der Dokumentenliste. '||
34                  'Bestätigen Sie durch Enter!';
35   LMESSAGE=160; TELLUSER(NEWLINE);
36  END SHOWLIST;
37
38  ERASESCREEN;
39  MESSAGELINE(1)=
40  'Geben Sie Dokumenten-Nummern ein (0 → Ende):';
41  LMESSAGE=80; TELLUSER(NEXTPOS);
42
43  MESSAGELAYOUT(ASKDOCUMENT),
44     2 TEXT_DOKUMENT CHAR(9),
45     2 NEWDOC PIC '999';
46  TEXT_DOKUMENT='Dokument='; LMESSAGE=9; LOFFER=3;
47
48  INITLIST(TOTALLIST); ASKNEXTDOC='1'B;
49  DO WHILE(ASKNEXTDOC);
50     NEWDOC='000'; ASKANDOFFERUSER(NEWLINE);
51     IF NEWDOC='000' /* Offerte akzeptiert */
52     THEN ASKNEXTDOC='0'B;
53     ELSE PUSHLIST(TOTALLIST,NEWDOC);
54  END;
55
56  MESSAGELINE(1)='Sie können nun die Dokumente '||
57                 'Anton und Berta zuweisen:';
58  LMESSAGE=80; TELLUSER(NEWPAGE);
59
60  MESSAGELAYOUT(ASKASSIGNMENT),
61     2 ASKPART,
62        3 TEXT_DOK CHAR(9),
63        3 ACTDOC PIC '999',
64     2 OFFERPART,
65        3 TEXT_FUER_ANTON CHAR(12),
66        3 ANTON_JA CHAR(1),
67        3 TEXT_FUER_BERTA CHAR(13),
68        3 BERTA_JA CHAR(1);
```

```
69  TEXT_DOK='Dokument=';
70  TEXT_FUER_ANTON=' für Anton: ';
71  TEXT_FUER_BERTA=', für Berta: ';
72  LMESSAGE=12; LOFFER=27;
73
74  REVERSELIST(TOTALLIST);
75  INITLIST(ANTONLIST); INITLIST(BERTALIST);
76  POPLIST(TOTALLIST,ACTDOC);
77  DO WHILE(¬EOF);
78     ANTON_JA='j'; BERTA_JA='j';
79     ASKANDOFFERUSER(NEWLINE);
80     IF ANTON_JA='j'
81     THEN PUSHLIST(ANTONLIST,ACTDOC);
82     IF BERTA_JA='j'
83     THEN PUSHLIST(BERTALIST,ACTDOC);
84     POPLIST(TOTALLIST,ACTDOC);
85  END;
86
87  CALL SHOWLIST('Anton',ANTONLIST);
88  RELEASELIST(ANTONLIST);
89  CALL SHOWLIST('Berta',BERTALIST);
90  RELEASELIST(BERTALIST);
91
92 %INCLUDE ROUTEND;
```

Anmerkungen zu den Programmzeilen:

2 Der Algorithmus `HANDLELISTS` wird als Routine vereinbart.

5 Drei Listen werden vereinbart, die die Nummern sämtlicher Dokumente bzw. der Dokumente für „Anton“ und „Berta“ aufnehmen sollen.

8-36 Die interne Prozedur `SHOWLIST` wird deklariert. Sie soll die Dokumentenlisten für die Benutzer „Anton“ bzw. „Berta“ ausgeben.

12 Ein Prozess `BROWSE` wird vereinbart, durch den die jeweilige Dokumentenliste durchlaufen werden soll.

16 Die Länge der Dokumentenliste wird ermittelt.

17-21 Am Beginn einer neuen Bildschirm-Seite wird auf zwei Zeilen die Anzahl der Dokumente in der Benutzerliste angezeigt.

23 Die Reihenfolge der Elemente in der Dokumentenliste wird ungedreht, damit diese im folgenden Listen-Browse-Prozess in der Reihenfolge ihrer Einspeicherung betreitgestellt werden.

24 Die Parameter für den Listen-Browse-Prozess werden vereinbart. Die Listenelemente sollen sukzessive in der Variablen `DOC_NR` bereitgestellt werden.

25 Ein Listen-Browse-Prozess wird generiert und sein Identifikator der Variablen `BROWSE` zugewiesen. Dabei wird das letzte Element der Dokumentenliste in der Variablen `DOC_NR` bereitgestellt. Ist die Liste jedoch leer, wird der Systemzustand `EOF` auf **true** (`'1'B`) gesetzt.

26-30 Für jedes Listenelement wird auf den jeweils nächsten 20 Bildschirm-Positionen die Dokumenten-Nummer ausgegeben und durch Aktivierung des Listen-Browse-Prozesses das vorangehende Listenelement bereitgestellt. Ist die Liste erschöpft, wird der Systemzustand `EOF` auf **true** (`'1'B`) gesetzt.

32-35 Auf zwei Zeilen wird der Benutzer darüber informiert, dass das Ende der Dokumentenliste erreicht ist.

38-90 Der Programmkörper der Prozedur `HANDLELISTS` wird angegeben.

38 Der Benutzer-Bildschirm wird gelöscht.

39-41 Eine Überschriftenzeile wird ausgegeben, die den Fachinformator dazu auffordert, im weiteren Verlauf Dokumenten-Nummern einzugeben, wobei die Dokumenten-Nummer `000` das Ende der Eingabe symbolisiert.

43-46 Eine Nachrichtenstruktur wird vereinbart, die aus der eigentlichen Nachricht (aus dem Text `'Dokument='`) und aus einer dreiziffrigen Offerte (`NEWDOC`) besteht.

48 Die Gesamtliste `TOTALLIST` wird initialisiert (leergesetzt).

49-54 Solange der Fachinformator die Eingabe von Dokumenten-Nummern nicht dadurch beendet, dass er die mit `000` vorbelegte Offerte `NEWDOC` akzeptiert, wird die von ihm eingegebene Dokumenten-Nummer in die Gesamtliste aufgenommen.

56-58 Auf einer neuen Bildschirm-Seite wird der Fachinformator in einer Zeile aufgefordert, seine eingegebenen Dokumenten-Nummern nun den Benutzern „Anton“ und/oder „Berta“ zuzuweisen.

60-68 Eine neue Nachrichtenstruktur wird vereinbart, deren Aufforderungs-Teil `ASKPART` die aktuelle Dokumenten-Nummer `ACTDOC` und deren Offerten-Teil `OFFERPART` die beiden Offerten `ANTON_JA` und `BERTA_JA` enthält.

69-72 Die konstanten Bestandteile der Nachrichtenstruktur werden belegt.

74 Die Reihenfolge der Elemente in der Gesamtliste wird umgekehrt, damit diese im folgenden in der Reihenfolge ihrer Einspeicherung bereitgestellt werden.

75 Die Dokumentenlisten für die Benutzer „Anton“ und „Berta“ werden initialisiert.

76 Die letzte Dokumenten-Nummer der Gesamtliste wird der Variablen `ACTDOC` zugewiesen und aus der Gesamtliste gelöscht.

77-85 Solange noch nicht alle Dokumenten-Nummern aus der Gesamtliste ausgelesen und dort gelöscht wurden, wird dem Fachinformator auf jeweils einer neuen Zeile die aktuelle Dokumenten-Nummer `ACTDOC` gemeinsam mit den Offerten `ANTON_JA` und `BERTA_JA`, die mit `'j'` vorbelegt sind, ausgegeben. Akzeptiert der Fachinformator eine oder beide Offerten, wird die Dokumenten-Nummer in die Dokumentenlisten eines oder beider Benutzer eingetragen.

87-90 Die Dokumentenlisten für die beiden Benutzer „Anton“ und „Berta“ werden durch die interne Prozedur `SHOWLIST` ausgegeben und dann gelöscht.

4.4.2 Zeichenketten

In Anwenderprogrammen ist es häufig erforderlich, Zeichenketten mit unterschiedlicher Länge abzuspeichern. KOMPROMISS bietet dafür das Prinzip der Halden-Organisation (Heap) an. Jede Zeichenkette belegt dabei auf dem Heap nur so viel Speicher, wie für ihre aktuelle Länge tatsächlich erforderlich ist. Wird eine Zeichenkette nicht mehr benötigt, sollte sie „abgemeldet“ werden, damit der dadurch freiwerdende Speicherbereich wieder für andere Zeichenketten genutzt werden kann.

Für die Speicherung sämtlicher Zeichenketten wird ein globaler Speicherbereich (Stringheap) benötigt, der durch die Anweisung

```
GETSTRINGHEAP(<Stringanzahl>,<mittlere-Länge>);
```

angefordert wird. Das erweiterte Steuerprogramm berechnet aus den Schätzwerten <Stringanzahl> und <mittlere-Länge> den bereitzustellenden Speicherbereich.

Der Zugriff auf eine Zeichenkette, die im Stringheap abgelegt ist, erfolgt über eine Referenz, die mit dem Datentyp `HEAPSTRING` zu vereinbaren ist:

```
DCL <HEAPSTRING-Variable> [<Dimension>] HEAPSTRING;
```

Vor der ersten Verwendung eines Heapstrings muss er initialisiert (leergesetzt) werden.

```
INITSTRING(<HEAPSTRING-Variable>);
```

Ob ein Heapstring leer ist, kann durch die folgende logische Funktion abgefragt werden:

```
IF   ISINITSTRING(<HEAPSTRING-Variable>)
THEN ...;
```

Alle Operationen mit Zeichenketten sind über PL/I-Anweisungen zu programmieren. KOMPROMISS unterstützt lediglich das „Abschicken“ einer Zeichenkette in den Stringheap und ihr „Zurückholen“.

Das „Abschicken“ einer Zeichenkette in den Stringheap erfolgt durch:

```
PUTCHARSTRING( { <CHAR-Variable>   } ,
               { '<Zeichenkette>'  }
                 <HEAPSTRING-Variable>) ;
```

Soll lediglich eine Teilzeichenkette im Stringheap abgelegt werden, so steht dafür das folgende Kommando zur Verfügung:

```
PUTSUBSTRING( <CHAR-Variable>,<Start-Position>,
              <Länge>,<HEAPSTRING-Variable>) ;
```

Findet die abgeschickte (Teil-)Zeichenkette Platz im Stringheap, erhält die <HEAPSTRING-Variable> deren Referenzwert. Lässt sie sich auch nach einem automatisch eingeleiteten Prozess der Speicherrückgewinnung („garbage collection“[1]) nicht im Stringheap unterbringen, wird die Aufgabenbearbeitung abnormal beendet.

Das „Zurückholen“ einer Zeichenkette aus dem Stringheap und das Zuweisen ihres Wertes an eine <CHAR-Variable> erfolgt durch die Anweisung:

[1] Vgl. Ernst (2003), S. 786.

```
GETCHARSTRING(<CHAR-Variable>,
              <HEAPSTRING-Variable>);
```

Soll der aus dem Heap zurückgeholte Wert in einen reservierten Bereich einer <CHAR-Variable> als Teilzeichenkette eingetragen werden, so geschieht das durch die Anweisung:

```
GETSUBSTRING(<CHAR-Variable>,<Start-Position>,
             <Längen-Variable>,
             <HEAPSTRING-Variable>);
```

Die <Längen-Variable> ist sowohl Input- als auch Output-Parameter. Ihr wird vor dem Kommando GETSUBSTRING die Länge des reservierten Bereichs zugewiesen. Der reservierte Bereich wird linksbündig mit der angeforderten Zeichenkette belegt. Ist diese zu lang, wird sie abgeschnitten. Ist sie kürzer als der reservierte Bereich, bleiben die restlichen Zeichen des reservierten Bereichs erhalten (kein Auffüllen mit Leerzeichen!) und die <Längen-Variable> wird mit der tatsächlichen Länge des Heapstrings belegt.

Wird eine Zeichenkette nicht mehr benötigt, sollte der durch sie im Stringheap belegte Speicherbereich durch die Anweisung

```
RELEASESTRING(<HEAPSTRING-Variable>);
```

freigegeben werden. Wird keine der im Stringheap gespeicherten Zeichenketten mehr benötigt, sollte der Stringheap durch die Anweisung

```
RELEASESTRINGHEAP;
```

wieder freigegeben werden.

Das folgende - etwas komplexere - Beispiel soll die Verwendung des Stringheaps sowie die Morphemanalyse veranschaulichen. Es soll außerdem die Programmierung eines Prozesses sowie die Kommunikation des KI-Assistenten mit dem Information-Retrieval-System STAIRS illustrieren.

Der KI-Assistent soll den Benutzer mit statistischen Informationen über alle Wörter versorgen, die in den Titeln der Dokumente seines Recherche-Ergebnisses vorkommen: Er soll erfahren, welche Wörter wie häufig in den Titeln der Ergebnis-Dokumente und wie häufig in der gesamten Datenbank auftreten.

Der KI-Assistent wird eingerichtet, indem der Benutzer während seines Dialogs mit dem Information-Retrieval-System STAIRS das Kommando

##STATISTICS

eingibt. Das erweiterte Steuerprogramm generiert und aktiviert eine Monitormaschine, die nach dem Algorithmus `STATISTICS` arbeitet:

```
1  /*********************************************/
2  STATISTICS: %INCLUDE ROUTINE;
3  /*********************************************/
4
5   DCL DICT_PROCESS PROCESS,
6       TITLE_RDR READER,
7       BROWSE_END BIT(1),
8       LINE CHAR(80),
9       ILINE BIN FIXED,
10      MAXLINE BIN FIXED,
11      DICT_WORD CHAR(31) VARYING,
12      DOC_FREQU PIC '999';
13
```

```
14  GETINTERFACE(39,DICT_INTERFACE);
15  DCLINTERFACE(DICT_INTERFACE),
16     2 SERVICE BIN FIXED,
17     2 NWORDS BIN FIXED,
18     2 WORD CHAR(31),
19     2 LWORD BIN FIXED,
20     2 WORD_FREQUENCY BIN FIXED;
21
22  SUBSTR(MESSAGE,1,16)='..SET NEWPAGE=ON';
23  LMESSAGE=16; ACTIVATE(IRS);
24
25  SUBSTR(MESSAGE,1,40)=
26  'Geben Sie die Nummer Ihrer Anfrage ein: ';
27  LMESSAGE=40; ASKUSER(NEWPAGE);
28
29  DO WHILE(¬MORPHEM(QNUMBER));
30     SUBSTR(MESSAGE,1,15)='Anfragenummer: ';
31     LMESSAGE=15; ASKUSER(NEWLINE);
32  END;
33
34  LMESSAGE=LINSTRING+12;
35  SUBSTR(MESSAGE,1,LMESSAGE)='..BROWSE '||
36                             INSTRING||' TI';
37  ACTIVATE(IRS);
38
39  MESSAGELAYOUT(DOCUMENT_PAGE),
40     2 HEADLINE,
41        3 DB_IDENTIFIER CHAR(12),
42        3 TEXT_DOCUMENT CHAR(10),
43        3 ACTDOC PIC '99999999',
44        3 TEXT_OF_1 CHAR(4),
45        3 LASTDOC PIC '99999999',
46        3 TEXT_PAGE CHAR(10),
47        3 ACTPAGE PIC '999999',
48        3 TEXT_OF_2 CHAR(4),
49        3 LASTPAGE PIC '999999',
50        3 HEADLINE_TAIL CHAR(12),
```

```
51     2 SEGMENTLINE(20),
52         3 SEGMENT_IDENTIFIKATOR CHAR(10),
53         3 TITLE_LINE CHAR(70);
54
55  GETSTRINGHEAP(LASTDOC*30,15);
56  DICT_INTERFACE.NWORDS=LASTDOC*30;
57  GENERATE('DICTIONARY',DICT_PROCESS);
58
59  DICT_INTERFACE.SERVICE=1; BROWSE_END='0'B;
60  DO WHILE(¬BROWSE_END);
61     ILINE=1; MAXLINE=(LMESSAGE/80)-1;
62     DO WHILE(ILINE<=MAXLINE &
63              TITLE_LINE(ILINE)¬=' ');
64
65        SCANCHARSTRING(TITLE_LINE(ILINE),70,
66                       TITLE-RDR);
67        DO WHILE(¬READERMORPHEM(TITLE_RDR,QEND));
68           IF READERMORPHEM(TITLE_RDR,QTAG) THEN
69           DO;
70               IF LINSTRING>31 THEN LINSTRING=31;
71               DICT_INTERFACE.WORD=INSTRING;
72               DICT_INTERFACE.LWORD=LINSTRING;
73               ACTIVATE(DICT_PROCESS);
74           END;
75           ELSE RELEASEREADERMORPHEM(TITLE_RDR);
76        END;
77        ILINE=ILINE+1;
78     END;
79
80     IF (ACTDOC=LASTDOC) & (ACTPAGE=LASTPAGE)
81     THEN BROWSE_END='1'B;
82     ELSE DO; LMESSAGE=0; ACTIVATE(IRS); END;
83  END;
84
85  SUBSTR(MESSAGE,1,8)='..SEARCH';
86  LMESSAGE=8; ACTIVATE(IRS);
87
```

```
88  MESSAGELINE(1)='Wörter aus den Dokumenten';
89  MESSAGELINE(2)='(mit Häufigkeit in den '||
90                 'Dokumenten und in der '||
91                 'ganzen Datenbank):';
92  MESSAGELINE(3)=' '; ILINE=3;
93  LMESSAGE=240; TELLUSER(NEWPAGE);
94
95  MESSAGELAYOUT(ROOT_RESULT),
96     2 SEARCH_MODE_LINE CHAR(80),
97     2 ROOT_COMMAND_LINE CHAR(80),
98     2 FREQUENCY_LINE,
99         3 ROOT_WORD CHAR(31),
100        3 DB_FREQUENCY PIC '99999999',
101        3 LINE_TAIL CHAR(41);
102
103 DICT_INTERFACE.SERVICE=2;
104 ACTIVATE(DICT_PROCESS);
105 DO WHILE(¬EOF);
106    DICT_WORD=SUBSTR(WORD,1,LWORD);
107    DOC_FREQU=WORD_FREQUENCY;
108    MESSAGELINE(1)='ROOT '||DICT_WORD||' 0';
109    LMESSAGE=LWORD+7; ACTIVATE(IRS);
110    ILINE=ILINE+1; LINE=DICT_WORD||': ';
111    SUBSTR(LINE,34,14)=DOC_FREQU||
112                       ' ('||DB_FREQUENCY||')';
113    MESSAGELINE(1)=LINE; LMESSAGE=80;
114    TELLUSER(NEWLINE);
115
116    IF ILINE=22 THEN
117    DO
118      MESSAGELINE(1)=' ';
119      MESSAGELINE(2)='Zum Übergang auf die '||
120                     'nächste Seite '||
121                     'Enter-Taste drücken!';
122      LMESSAGE=160; ASKUSER(NEWLINE);
123
```

```
124        MESSAGELINE(1)=
125        'Wörter aus den Dokumenten (Forts.):';
126        MESSAGELINE(2)=' '; ILINE=2;
127        LMESSAGE=160; TELLUSER(NEWPAGE);
128      END;
129      ACTIVATE(DICT_PROCESS);
130  END;
131
132  MESSAGELINE(1)=' ';
133  MESSAGELINE(2)='Ende der Wortliste';
134  LMESSAGE=160; TELLUSER(NEWLINE);
135
136  RELEASEINTERFACE(DICT_INTERFACE);
137  RELEASESTRINGHEAP;
138  TERMINATE(DICT_PROCESS);
139
140  MESSAGELINE(1)=
141  'Jetzt arbeiten Sie wieder mit STAIRS.';
142  MESSAGELINE(2)=
143  'Geben Sie ein STAIRS-Kommando ein!';
144  LMESSAGE=160;
145  LASTWILLFORUSER(NEWPAGE);
146
147 %INCLUDE ROUTEND;
```

Anmerkungen zu den Programmzeilen:

2 Der Algorithmus `STATISTICS` wird als Routine vereinbart.

5 Die `PROCESS`-Variable `DICT_PROCESS` soll den Identifikator der Wörterbuch-Maschine aufnehmen.

6 Der Lesekopf (`READER`-Variable) `TITLE_RDR` soll die lexikalische Analyse für eine Zeile des Dokumenten-Titels ausführen.

11 Die String-Variable `DICT_WORD` soll ein Wort aufnehmen, das aus dem Wörterbuch bereitgestellt wird. Sie ist auf eine Länge von 31 Zeichen begrenzt, weil dies der Maximallänge der Wörter entspricht, die durch STAIRS recherchiert werden können.

14-20 Das Interface `DICT_INTERFACE` wird angefordert und in seiner Struktur beschrieben. Es dient zur Kommunikation zwischen der Monitor-Maschine `STATISTICS` und der Wörterbuch-Maschine `DICTIONARY`.

16-17 Die Variable `SERVICE` gibt den Dienst an, der jeweils von der Wörterbuch-Maschine `DICTIONARY` angefordert wird. `NWORDS` ist die Maximalanzahl der Wörter, die das Wörterbuch insgesamt verwalten soll.

18-20 `WORD_FREQUENCY` ist die Häufigkeit, mit der ein Wort `WORD` mit seiner Länge `LWORD` in den Dokumenten des Recherche-Ergebnisses vorkommt.

22-23 Mit dem Kommando `'..SET NEWPAGE=ON'` wird STAIRS angewiesen, im nachfolgenden BROWSE-Prozess jedes Dokument ab dem Beginn einer neuen Seite auszugeben. Das bietet nämlich den Vorteil, dass sich für die anschließende Analyse die Struktur einer vom Information-Retrieval-System STAIRS ausgegebenen Seite durch `MESSAGELAYOUT(DOCUMENT_PAGE)` beschreiben lässt.

25-27 Der Benutzer wird auf einer neuen Bildschirm-Seite dazu aufgefordert, die Referenznummer seiner Suchanfrage, für die er eine Wortstatistik wünscht, einzugeben.

29-32 Wenn seine Antwort nicht mit einer Zahl beginnt, wird auf jeweils einer neuen Zeile solange nachgefragt, bis er der Aufforderung nachkommt und eine Anfragenummer eingibt.

34-37 An STAIRS wird ein BROWSE-Kommando mit der Aufforderung abgeschickt, für alle Ergebnis-Dokumente der angegebenen Suchanfrage die Titel auszugeben („TI“ ist der Segment-Identifikator des Dokumententitels). Als Antwort wird die erste Seite des ersten Dokuments bereitgestellt.

39-53 DOCUMENT_PAGE beschreibt die Struktur einer im BROWSE-Prozess ausgegebenen Seite. In der Überschriftenzeile HEADLINE werden neben dem Identifikator der Datenbank die aktuelle und die letzte Dokumenten-Nummer des Recherche-Ergebnisses sowie die Nummern der ersten und der letzten Seite des aktuellen Dokuments angegeben. Die sich daran anschließenden maximal 20 Zeilen SEGMENTLINE(20) enthalten jeweils in TITLE_LINE den Text des Dokumenten-Titels.

55-57 Mit der ersten Seite des ersten Dokuments steht fest, wie viele Dokumente das Recherche-Ergebnis enthält (LASTDOC). Jetzt kann der Stringheap für die Speicherung von durchschnittlich 30 Wörtern je Dokumenten-Titel mit einer mittleren Wortlänge von 15 Zeichen angefordert werden. Außerdem wird der DICT_PROCESS nach dem Algorithmus DICTIONARY für durchschnittlich 30 Wörter je Dokumenten-Titel generiert.

59-83 Sämtliche im BROWSE-Prozess bereitgestellten Seiten der Ergebnis-Dokumente werden analysiert.

59 Im Interface DICT_INTERFACE für die Kommunikation mit der Wörterbuch-Maschine wird der angeforderte Dienst durch die Zuweisung SERVICE=1 („Berücksichtigung“ eines Wortes) angegeben.

61-78 Alle Titelzeilen einer Seite – ab der ersten Zeile (ILINE=1) bis zur letzten nichtleeren Zeile – werden analysiert. Bei der Berechnung von MAXLINE wird berücksichtigt, dass sich im Nachrichtenbereich MESSAGE auch die Überschriftenzeile HEADLINE befindet.

65-66 Der Lesekopf TITLE_RDR wird für die lexikalische Analyse der aktuellen Titelzeile initialisiert.

67-76 Durch den Lesekopf wird die Zeile bis an ihr Ende (Morphemklasse QEND) analysiert.

68-74 Stellt der Lesekopf ein Morphem der Klasse QTAG in der Systemvariablen INSTRING mit der Länge LINSTRING bereit, so wird die Wortlänge zunächst auf 31 begrenzt (s. Kommentar zur Programmzeile 10). Dann wird das Wort in das Interface DICT_INTERFACE

eingetragen und der Wörterbuch-Prozess aktiviert. Dadurch wird das Wort im Wörterbuch „berücksichtigt“ – also entweder als neues Wort gespeichert oder seine Häufigkeit um eins erhöht.

75 Handelt es sich bei dem aktuellen Morphem nicht um ein Wort, so wird es überlesen.

80-82 Wurde bereits die letzte Seite des letzten Ergebnis-Dokuments ausgewertet, dann wird der BROWSE-Prozess beendet. Ansonsten wird durch Absenden einer leeren Nachricht (das entspricht dem einfachen Betätigen der Enter-Taste) STAIRS zur Bereitstellung der nächsten Seite aufgefordert.

85-86 Das IRS STAIRS wird in den SEARCH-Modus (Recherche-Modus) überführt, damit es im folgenden die ROOT-Kommandos akzeptiert.

88-93 Am Beginn einer neuen Seite wird auf drei Zeilen der Kopf der Wortstatistik ausgegeben.

95-101 `ROOT_RESULT` beschreibt die Struktur der Ausgabe, mit der STAIRS auf ein ROOT-Kommando reagiert. Durch ein solches Kommando wird die Häufigkeit eines Wortes bzw. eines Wortstammes in der gesamten Datenbank abgefragt. Die erste Zeile enthält die Angabe, dass sich STAIRS im SEARCH-Modus (Recherche-Modus) befindet, die zweite Zeile wiederholt das eingegebene ROOT-Kommando, die dritte Zeile enthält mit `DB_FREQUENCY` die achtstellige Häufigkeit des Wortes in der Datenbank.

103-104 Im Interface `DICT_INTERFACE` für die Kommunikation mit der Wörterbuch-Maschine wird der angeforderte Dienst durch die Belegung `SERVICE=2` (Bereitstellen aller Wörter) angegeben und der Wörterbuch-Prozess aktiviert. Durch diese Aktivierung wird das lexikalisch kleinste Wort (einschließlich seiner Häufigkeit) bereitgestellt.

105-130 Solange der Wort-Bereitstellungszyklus noch nicht sein Ende erreicht hat (dann wird die Systemvariable `EOF` auf **true** (`'1'B`) gesetzt), werden alle in lexikalischer Folge bereitgestellten Wörter gemeinsam mit ihren Häufigkeiten auf dem Bildschirm des Benutzers ausgegeben.

106-107 Das im Interface DICT_INTERFACE bereitgestellte Wort WORD mit seiner Länge LWORD und seiner Häufigkeit in den Dokumenten des Recherche-Ergebnisses (WORD_FREQUENCY) wird in lokalen Variablen gespeichert.

108-109 An STAIRS wird ein ROOT-Kommando abgeschickt, durch das die Häufigkeit des Wortes DICT_WORD in der Datenbank abgefragt wird. Die Null nach dem DICT_WORD besagt dabei, dass sich an die angegebene Wortform ***keine*** weiteren Zeichen mehr anschließen dürfen. Es wird also exakt nach dem angegebenen Wort gesucht und dieses nicht als Wortstamm (Truncation-Wort) behandelt, das um weitere Zeichen erweitert werden kann.

110-114 Der Antwort von STAIRS mit der Struktur ROOT_RESULT wird die Anzahl DB_FREQUENCY entnommen und gemeinsam mit dem Wort (DICT_WORD) und dessen Häufigkeit in den Ergebnis-Dokumenten (DOC_FREQU) auf der nächsten Bildschirm-Zeile ausgegeben.

116-128 Wurden auf der aktuellen Bildschirm-Seite schon 22 Zeilen ausgegeben, so werden nun die letzten beiden Zeilen dieser Seite ausgegeben, die den Benutzer dazu auffordern, durch Betätigen der Enter-Taste den Aufbau der nächsten Bildschirm-Seite auszulösen, die dann den zweizeiligen Kopf einer Folgeseite enthält.

129 Der Wörterbuch-Prozess wird erneut aktiviert, damit er das nächste Wort bzw. das Ende der Wortbereitstellung (EOF='1'B) bereitstellt.

132-134 Da auf der aktuellen Bildschirm-Seite noch wenigstens zwei Zeilen frei sind, kann dem Benutzer auf zwei Zeilen das Ende der Wortliste mitgeteilt werden.

136 Das Interface DICT_INTERFACE für die Kommunikation mit der Wörterbuch-Maschine wird nicht mehr benötigt und kann deshalb freigegeben werden.

137 Der nicht mehr benötigte Stringheap wird freigegeben.

138 Die Wörterbuch-Maschine, die auf Grund einer Endlosschleife in der Dienste-Erbringung ihren Endpunkt nicht von alleine erreichen kann, wird zwangsweise aus dem KI-Assistenten entfernt.

140-145 Das „Testament“ der Monitormaschine mit der Länge `LMESSAGE` wird in zwei Zeilen des Nachrichtenbereichs `MESSAGE` hinterlegt und auf einer neuen Bildschirm-Seite ausgegeben. Der KI-Assistent wird daraufhin abgebaut, und der Benutzer tritt wieder in die direkte Kommunikation mit dem Information-Retrieval-System STAIRS.

Die Monitor-Maschine zum Algorithmus `STATISTICS` löst die ihr zugedachte Aufgabe in Kooperation mit einer Wörterbuch-Maschine, die nach dem Algorithmus `DICTIONARY` arbeitet. Dieser Algorithmus ist so beschaffen, dass er die zu erbringenden Ergebnisse in kleinen Portionen bereitstellt – er ist somit vom Typ eines Prozesses. Die Wörterbuch-Maschine erbringt im Sinne eines „Servers“ Dienste für andere „Clients“ (im vorliegenden Beispiel lediglich für die Monitor-Maschine).

Der Algorithmus `DICTIONARY` besteht aus zwei Teilen: aus dem Initialisierungs-Teil, in dem Anfangszustände gesetzt werden, und aus dem Service-Teil, in dem die eigentlichen Dienste erbracht werden. Dabei werden in der angegebenen Ausbaustufe des Algorithmus' lediglich zwei Dienste angeboten:

1. ***„Berücksichtigung“ eines Wortes:*** Ist das Wort neu, wird es in das Wörterbuch aufgenommen und seine Häufigkeit auf eins gesetzt. Ist es bereits gespeichert, wird seine Häufigkeit um eins erhöht.

2. ***Bereitstellen aller Wörter:*** Die Wörter werden (gemeinsam mit ihrer Häufigkeit) sukzessive in lexikalischer Reihenfolge bereitgestellt.

Im folgenden wird der Algorithmus `DICTIONARY` der Wörterbuch-Maschine angegeben:

```
1  /*********************************************/
2  DICTIONARY: %INCLUDE PROCESS;
3  /*********************************************/
4
5   USEINTERFACE(DICT_INTERFACE);
6   DCLINTERFACE(DICT_INTERFACE),
7      2 SERVICE BIN FIXED,
8      2 NWORDS BIN FIXED,
9      2 WORD CHAR(31),
10     2 LWORD BIN FIXED,
11     2 WORD_FREQUENCY BIN FIXED;
12
13  DCL MAX_WORDNUMBER BIN FIXED;
14  MAX_WORDNUMBER=NWORDS;
15
16  BEGIN /* DYNAMIC DECLARATION BLOCK */;
17
18  DCL NODE(MAX_WORDNUMBER,3) BIN FIXED,
19      WORD_ARRAY(MAX_WORDNUMBER) HEAPSTRING,
20      LESS_TREE BIN FIXED INIT(1),
21      GREATER_TREE BIN FIXED INIT(2),
22      FREQUENCY BIN FIXED INIT(3),
23      ROOT BIN FIXED,
24      INODE BIN FIXED,
25      LOOKUP_WORD CHAR(31) VARYING;
26
27  DCL NEWWORD ENTRY RETURNS(BIN FIXED);
28  NEWWORD: PROCEDURE RETURNS(BIN FIXED);
29   IF INODE=MAX_WORDNUMBER
30   THEN ABEND('Fehler: Wörterbuch zu klein') ELSE
31   DO;
32      INODE=INODE+1;
33      NODE(INODE,LESS_TREE)=0;
34      NODE(INODE,GREATER_TREE)=0;
35      NODE(INODE,FREQUENCY)=1;
36      INITSTRING(WORD_ARRAY(INODE));
37      PUTCHARSTRING(LOOKUP_WORD,
38                    WORD_ARRAY(INODE));
```

```
39        RETURN(INODE);
40      END;
41    END NEWWORD;
42
43    DCL LOOKUP ENTRY(BIN FIXED) RECURSIVE;
44    LOOKUP: PROCEDURE(FATHER) RECURSIVE;
45      DCL FATHER BIN FIXED;
46      DCL FATHER_WORD CHAR(31) VARYING,
47          SONPOS BIN FIXED,
48          SON BIN FIXED;
49
50      GETCHARSTRING(FATHER_WORD,
51                    WORD_ARRAY(FATHER));
52
53      IF LOOKUP_WORD=FATHER_WORD
54      THEN NODE(FATHER,FREQENCY)=
55           NODE(FATHER,FREQENCY)+1;
56      ELSE
57      DO;
58           IF LOOKUP_WORD<FATHER_WORD
59           THEN SONPOS=LESS_TREE;
60           ELSE SONPOS=GREATER_TREE;
61           SON=NODE(FATHER,SONPOS);
62           IF SON=0
63           THEN NODE(FATHER,SONPOS)=NEWWORD;
64           ELSE CALL LOOKUP(SON);
65      END;
66    END LOOKUP;
67
68    DCL TRAVERSE ENTRY(BIN FIXED) RECURSIVE;
69    TRAVERSE: PROCEDURE(FATHER) RECURSIVE;
70      DCL FATHER BIN FIXED;
71      DCL SON BIN FIXED;
72
73      SON=NODE(FATHER,LESS_TREE);
74      IF SON¬=0 THEN CALL TRAVERSE(SON);
75
```

```
76     LWORD=31;
77     GETSUBSTRING(WORD,1,LWORD,
78                  WORD_ARRAY(FATHER));
79     WORD_FREQUENCY=NODE(FATHER,FREQENCY);
80     EOF='0'B; READY;
81
82     SON=NODE(FATHER,GREATER_TREE);
83     IF SON¬=0 THEN CALL TRAVERSE(SON);
84   END TRAVERSE;
85
86   ROOT=0; INODE=0; READY; /* INITIALISATION */
87
88   DO WHILE('1'B); /* SERVICES */
89
90      IF SERVICE=1 /* CONSIDER A WORD */ THEN
91      DO;
92         LOOKUP_WORD=SUBSTR(WORD,1,LWORD);
93         IF ROOT=0 THEN ROOT=NEWWORD;
94         ELSE CALL LOOKUP(ROOT);
95         READY;
96      END /* SERVICE=1 */ ELSE
97
98      IF SERVICE=2 /* TRAVERSE DICTIONARY */ THEN
99      DO;
100        IF ROOT¬=0 THEN CALL TRAVERSE(ROOT);
101        EOF='1'B; READY;
102     END /* SERVICE=2 */
103
104  END /* SERVICES */;
105
106  END /* DYNAMIC DECLARATION BLOCK */;
107
108 %INCLUDE PROCEND;
```

Anmerkungen zu den Programmzeilen:

2 Der Algorithmus DICTIONARY wird als Prozess vereinbart.

5-11 Das bereits existierende Interface DICT_INTERFACE (es wurde von der Monitor-Maschine angelegt) ist für den Datenaustausch mit dem jeweiligen Client zuständig. Durch USEINTERFACE wird der Zugriff auf die Interface-Daten ermöglicht, deren Struktur im weiteren beschrieben wird.[1]

13-14 Die Variable MAX_WORDNUMBER wird deklariert und mit der angeforderten maximalen Wortanzahl NWORDS belegt. Das „Retten" dieser Größe in einer lokalen Variablen ist erforderlich, weil das Interface DICT_INTERFACE im folgenden dazu verwendet werden kann, um weitere Maschinen zum Algorithmus DICTIONARY einzurichten, die jeweils ihre eigene maximale Wortanzahl haben.

16-106 Für die Einrichtung der beiden Felder NODE und WORD_ARRAY mit ihrer dynamischen Feldgrenze MAX_WORDNUMBER ist ein BEGIN-END-Block erforderlich.

18-19 Das Feld NODE dient zur Darstellung eines binären Baums. In WORD_ARRAY werden die Wörter als HEAPSTRING-Variablen abgelegt.

20-22 Die drei Komponenten eines Knotens im Feld NODE haben die folgende Bedeutung: NODE(<Index>,LESS_TREE) referiert einen Teilbaum mit Wörtern, die kleiner sind als das Wort, das dem Knoten entspricht. NODE(<Index>,GREATER_TREE) referiert einen Teilbaum mit größeren Wörtern. NODE(<Index>,FREQUENCY) enthält die Häufigkeit, mit der das Wort dem Wörterbuch angeboten wurde.

[1] Die Beschreibung einer Interface-Struktur muss in exakt übereinstimmender Form in sämtliche Programme eingefügt werden, die das Interface benutzen. Damit die Konsistenz der Datenbeschreibung gewahrt bleibt, sollte es nur einmal unter einem Namen <Interface-Name> gespeichert und dann in alle Programme durch die Anweisung '%INCLUDE <Interface-Name>' eingefügt werden.

23-24 `ROOT` ist die Wurzel des binären Baums, `INODE` der laufende Wortindex.

27-41 Die interne Funktion `NEWWORD` übernimmt die Speicherung eines neuen Wortes in Form eines weiteren Baum-Knotens. Sie liefert den Index des neuen Knotens.

29-30 Sollen mehr Wörter in das Wörterbuch eingespeichert werden als durch den Generierungs-Parameter `MAX_WORDNUMBER` vorgesehen sind, wird die Arbeit des KI-Assistenten abnormal beendet. Der angegebene Text wird dem Benutzer angezeigt.

31-40 Das neue Wort wird als weiterer Baum-Knoten gespeichert. Sowohl sein kleinerer als auch sein größerer Teilbaum werden leergesetzt (das wird durch die Referenz `0` symbolisiert). Die Häufigkeit wird auf eins gesetzt. Das `HEAPSTRING`-Element `WORD_ARRAY(INODE)` wird initialisiert und mit der Referenz auf das in den Stringheap abgeschickte `LOOKUP_WORD` belegt. Der Index des neuen Knotens wird als Funktionswert zurückgegeben.

43-66 Die interne rekursive Prozedur `LOOKUP` sucht im Wörterbuch-Baum nach dem `LOOKUP_WORD`. Ist es bereits gespeichert, wird seine Häufigkeit um eins erhöht, ansonsten wird es als neues Wort mit der Häufigkeit eins in den Wörterbuch-Baum aufgenommen.

45 Der Parameter `FATHER` ist der Index des Vater-Knotens des zu analysierenden Baums[1].

50-51 Das im Stringheap gespeicherte Wort, das dem Vater-Knoten zugeordnet ist, wird der lokalen Variablen `FATHER_WORD` zugewiesen.

53-55 Stimmt das aufzusuchende Wort mit dem Wort des Vater-Knotens überein, wird dessen Häufigkeit um eins erhöht.

56-65 Andernfalls muss in tieferen Zweigen des Wörterbuch-Baums nach dem Wort gesucht werden.

---

[1] Genaugenommen ist FATHER der Index des ***Stammvater***-Knotens für alle Söhne, Enkel, Urenkel usw., die ihm im Wörterbuch-Baum untergeordnet sind.

58-61 In der lokalen Variablen SONPOS wird festgehalten, ob weiter im Teilbaum für kleinere Wörter LESS_TREE oder im Teilbaum für größere Wörter GREATER_TREE zu suchen ist. Entsprechend wird der Teilbaum SON ausgewählt.

62-63 Gibt es den Teilbaum SON - und damit auch das gesuchte Wort - noch nicht, wird das Wort durch die Funktion NEWWORD als neuer Knoten angelegt, der beim Vater-Knoten FATHER in der entsprechenden Sohn-Position SONPOS „eingehängt“ wird.

64 Wenn der Teilbaum SON vorhanden ist, dann wird er nach dem LOOKUP_WORD durchsucht.

68-84 Die interne rekursive Prozedur TRAVERSE soll einen Wörterbuch-Baum vollständig durchlaufen und die in ihm enthaltenen Wörter (einschließlich ihrer Häufigkeit) in lexikalischer Reihenfolge bereitstellen.

73-74 Wenn es den Teilbaum SON=NODE(FATHER,LESS_TREE) mit Wörtern gibt, die kleiner sind als das dem Vater-Knoten FATHER zugeordnete Wort, dann werden diese durch einen rekursiven Aufruf von TRAVERSE bereitgestellt.

76-79 Das dem Vater-Knoten zugeordnete Wort wird aus dem Stringheap zurückgeholt und in das Interface DICT_INTERFACE als WORD mit der Länge LWORD eingetragen. Dafür werden zunächst alle 31 Zeichenpositionen in WORD zur Verfügung gestellt. Diese Maximallänge wird von GETSUBSTRING in LWORD auf die tatsächliche Länge des Wortes korrigiert. Die Häufigkeit, mit der das Wort dem Wörterbuch angeboten wurde, wird ebenfalls in das Interface DICT_INTERFACE (in die Variable WORD_FREQUENCY) eingetragen.

80 Die Steuerung wird an den Aktivator mit der Information zurückgegeben, dass der Wörterbuch-Baum noch nicht vollständig durchlaufen wurde (EOF='0'B).

82-83 Wenn es den Teilbaum SON=NODE(FATHER,GREATER_TREE) mit Wörtern gibt, die größer sind als das dem Vater-Knoten FATHER

zugeordnete Wort, dann werden diese durch einen rekursiven Aufruf von `TRAVERSE` bereitgestellt.

86 Der Initialisierungs-Teil wird im Zuge der Generierung der Wörterbuch-Maschine ausgeführt. Dabei wird der Wörterbuch-Baum leergesetzt (`ROOT=0`) und der laufende Knoten-Index auf Null gesetzt. Dann wird die Steuerung an den Generator zurückgegeben. Die Wörterbuch-Maschine steht nun für beliebige Clients bereit, um die angebotenen Dienstleistungen zu erbringen.

88-104 Der Service-Teil ist als Endlos-Schleife programmiert, so dass der Algorithmus nie seinen Endpunkt erreicht: Solange die Wörterbuch-Maschine nicht „gewaltsam" aus dem KI-Assistenten entfernt wurde, steht sie den Clients mit ihren Dienstleistungen zur Verfügung.

90-96 Durch den ersten Dienst (`SERVICE=1`) wird ein angebotenes Wort „berücksichtigt", d.h. es wird entweder als neues Wort in den Wörterbuch-Baum aufgenommen oder seine Häufigkeit wird um eins erhöht.

92 Das Wort, das im Interface `DICT_INTERFACE` übergeben wurde, wird der Variablen `LOOKUP_WORD` zugewiesen und steht nun in den Unterprogrammen `NEWWORD` und `LOOKUP` zur Verfügung.

93-95 Ist der Wörterbuch-Baum noch leer (`ROOT=0`), wird er als Knoten für das erste Wort `LOOKUP_WORD` gebildet. Ansonsten wird im gesamten Wörterbuch-Baum nach dem Wort gesucht. Ist es dort noch nicht vorhanden, wird es als neues Wort in den Wörterbuch-Baum aufgenommen. Die Steuerung wird an den Aktivator zurückgegeben.

98-102 Durch den zweiten Dienst (`SERVICE=2`) werden alle Wörter des Wörterbuch-Baums (gemeinsam mit ihrer Häufigkeit) sukzessive in lexikalischer Reihenfolge bereitgestellt.

100-101 Wenn der Wörterbuch-Baum nicht leer ist (`ROOT¬=0`), wird er durch die rekursive Prozedur `TRAVERSE` durchlaufen, wobei sukzessive alle Wörter bereitgestellt werden. Gemeinsam mit der Information, dass keine weiteres Wort mehr bereitgestellt werden kann (`EOF='1'B`), wird die Steuerung an den Aktivator zurückgegeben.

4.5 Implementierung der Programmiersprache

KOMPROMISS verwendet als Wirtssprache die Programmiersprache PL/I. Damit ergibt sich die Möglichkeit, in KOMPROMISS-Programmen alle in PL/I verfügbaren Datentypen zu vereinbaren und zu verarbeiten.

Alle Variablen des Programms - seien es nun Variable mit PL/I-Datenattributen oder KOMPROMISS-Datenattributen - bilden das Datenobjekt, das von der jeweiligen Maschine des KI-Assistenten bearbeitet wird. Durch die Nutzung des in PL/I verfügbaren Blockkonzepts können die Datenobjekte zeitweilig vergrößert werden. Eine solche zeitweilige Vergrößerung des Datenobjekts erfolgt ebenso bei Verwendung interner Prozeduren, die auch rekursiv aufgerufen werden können.

Die Programm-Logik wird durch die PL/I-Anweisungen für Verzweigungen und Befehlsschleifen ausgedrückt. Bei der Implementierung von KOMPROMISS war zu sichern, dass die Programme reenterabel werden, damit sie unter Steuerung des Transaktions-Managers CICS von vielen Benutzern quasiparallel genutzt werden können. Aus dieser Forderung ergeben sich einige Einschränkungen hinsichtlich der Verwendbarkeit der Sprachelemente von PL/I. Insbesondere dürfen die Sprachkonstruktionen von PL/I zur Arbeit mit Files nicht verwendet werden. Deshalb stellt KOMPROMISS eigene Anweisungen bereit, die den Zugriff auf sequentielle, indexsequentielle und direkt organisierte Dateien ermöglichen.

Für das Testen von Anwenderprogrammen bietet KOMPROMISS Online-Testhilfen an, die zu beliebigen Zeitpunkten in Kraft gesetzt und wieder außer Kraft gesetzt werden können. Sie betreffen insbesondere folgende Möglichkeiten:

1. Ausgabe eines Protokolls der Ein- und Ausgaben, die über den Bildschirm des Benutzers laufen,

2. Aufzeichnung der Steuerungsübergaben, die in den Sendepunkten der Maschinen des KI-Assistenten erfolgen (tracing),

3. Aufzeichnung des Nachrichtenaustausches, der – für den Benutzer unsichtbar - zwischen den Maschinen des KI-Assistenten einerseits und der Maschine 'IRS' andererseits abläuft.

Die Programmiersprache KOMPROMISS wurde ursprünglich für das Information-Retrieval-System STAIRS als Musterlösung implementiert. Die Sprache KOMPROMISS kann jedoch auch für die Programmierung eines interaktiven Programmsystems verwendet werden, das völlig ohne ein Information-Retrieval-System arbeitet.

5 Entwurf eines KI-Assistenten

In diesem Kapitel wenden wir uns der Frage zu, wie Anwenderprogramme zur Realisierung von KI-Assistenten für das Information-Retrieval-System STAIRS entworfen werden. Die Grundlage dafür bilden das im Kapitel 3 entwickelte Modell der abstrakten Maschinen und die im Kapitel 4 beschriebenen Ausdrucksmittel der Programmiersprache KOMPROMISS. Wir wollen zeigen, wie der übliche Zyklus des Software-Entwurfs in der Programmier-Umgebung von KOMPROMISS gestaltet werden kann.

Entsprechend dem allgemeinen Phasenmodell des Software-Entwurfs beginnt der Entwurfs-Zyklus in einer Analyse-Phase mit der Erarbeitung eines theoretischen Modells der Aufgabenlösung.[1] Daran schließt sich eine Konzeptions-Phase an, in der eine zweckmäßige Architektur von Maschinensystemen ermittelt wird. In der Realisierungs-Phase werden schließlich die Algorithmen entworfen, nach denen die Maschinen des KI-Assistenten arbeiten sollen.

Diesen Entwurfszyklus werden wir am Beispiel eines KI-Assistenten durchlaufen. Dabei werden wir uns bemühen, verallgemeinerungswürdige methodische Konzepte sichtbar zu machen.

5.1 Aufgabenstellung für den KI-Assistenten

Die Hauptfunktion des Information-Retrieval-Systems STAIRS besteht darin, dem Benutzer Dokumente bereitzustellen, die seinem Informationsbedarf entsprechen. Dazu muss der Benutzer seinen Informationsbedarf in Form einer Suchanfrage an das IRS formulieren. Das ist insbesondere bei der Recherche in Datenbanken ohne Thesaurus eine komplizierte Aufgabe, für die STAIRS kaum Hilfsmittel bereitstellt.

[1] Vgl. beispielsweise Abts/Mülder (2002), S. 17.

Das Problem, den Benutzer bei seiner Kommunikation mit einem Information-Retrieval-System zu unterstützen, gewinnt ständig an Bedeutung. Während in den siebziger Jahren noch etwa 90% der Benutzer von Informationssystemen Fachinformatoren waren, nahm in den letzten Jahrzehnten die Anzahl jener Benutzer stark zu, die nicht speziell für die Arbeit mit Information-Retrieval-Systemen ausgebildet sind. Schon 1984 betrug der Anteil der Fachinformatoren an den registrierten Benutzern des Informationsdienstes Dialog[1] nur noch 33%. Mit der breiten Nutzung des Internets, das den Zugang zu einer wachsenden Anzahl bibliographischer Datenbanken ermöglicht, verschiebt sich dieses Verhältnis ständig weiter zugunsten der ungeübten Benutzer. Daher sind alle Anbieter bibliographischer Datenbanken bestrebt, ihre Systeme mit Benutzeroberflächen auszustatten, die dem unerfahrenen Benutzer die Arbeit erleichtern sollen.[2]

Bei der Formulierung seiner Suchanfrage steht der Benutzer zunächst vor dem Problem, die seinen Informationsbedarf kennzeichnenden Begriffe durch solche Wörter und Wortkombinationen (Kollokationen[3]) auszudrücken, deren Auftreten in einem Dokument als Indiz dafür gewertet werden kann, dass im Dokument von diesem Begriff die Rede ist. Wenn ihm das gelungen ist, muss er durch logische Operatoren und Kontextoperatoren jene Konstrukte, die das Auftreten von Begriffen abfragen, schrittweise derart zu einer Gesamt-Suchanfrage verknüpfen, dass das Recherche-Ergebnis ausreichend vollständig und genau ist. In diesem Prozess treten Situationen ein, in denen der Benutzer nach der Trial-and-error-Methode verschiedene Alternativen erproben möchte. Er kann den Wunsch haben, entweder die Begriffskonstrukte zu verändern oder diese in anderer Weise miteinander zu verknüpfen.

Wir stellen uns die Aufgabe, einen KI-Assistenten zu entwerfen, der dem ungeübten Benutzer Hilfestellungen bei der Konstruktion seiner Suchanfrage bietet:

1. Durch eine geeignete Menüführung soll der Benutzer dazu angehalten werden, seine Suchanfrage nach einer semantisch orientierten Methode zu konstruieren.

2. Mittels statistischer Verfahren sollen Wörter aus Dokumenten ausgewählt und dem Benutzer zur Aufnahme in seine Begriffskonstrukte angeboten werden.

---

[1] Vgl. http://www.dialog.com.
[2] Vgl. beispielsweise Brajnik/Mizzaro/Tasso (2002).
[3] Vgl. beispielhaft Mittendorf/Mateev/Schäuble (2000).

3. Unter Verwendung von Methoden des Begriffslernens soll der Benutzer dazu angeleitet werden, seine Begriffskonstrukte derart zu einer Gesamt-Suchanfrage zu verknüpfen, dass akzeptable Recherche-Ergebnisse erzielt werden.

Aus der Gesamtaufgabe des KI-Assistenten wollen wir einzelne Teilaufgaben ableiten und dafür Teilsysteme von abstrakten Maschinen entwerfen. Im Rahmen dieses Buchs können wir nur einige dieser Teilsysteme beschreiben. Wir beschränken uns dabei auf jene Teilsysteme, in denen interessante methodische Konzepte realisiert werden. Teilsysteme, die hauptsächlich organisatorische Aufgaben - beispielsweise das Erfassen und Korrigieren der Suchanfrage - lösen, werden nicht beschrieben.

Jeder KI-Assistent besitzt eine Monitormaschine, die den gesamten Informationsverarbeitungsprozess steuert. Die Monitormaschine zur Realisierung unserer Aufgabenstellung nennen wir 'RECHERCHE'. Im folgenden Abschnitt beschreiben wir zunächst die Monitormaschine und präzisieren dann schrittweise Funktion und Struktur des KI-Assistenten.

5.2 Die Monitormaschine 'RECHERCHE'

Durch den KI-Assistenten soll der Benutzer dabei unterstützt werden, Suchanfragen so zu formulieren, dass sie möglichst exakt seinem Informationsbedarf entsprechen. Der Gesamtprozess der Erarbeitung einer Suchanfrage wird dabei in zwei Phasen unterteilt:

1. **Erfassungsphase**: Sie beinhaltet alle Aktivitäten, die mit dem Erfassen, dem eventuellen Korrigieren und der Syntaxprüfung der Suchanfrage zusammenhängen.

2. **Präzisierungsphase**: In dieser Phase unterstützt der KI-Assistent den Benutzer dabei, seine Suchanfrage zu präzisieren.

Die Monitormaschine 'RECHERCHE' übernimmt die Steuerung des Gesamtprozesses zur Erarbeitung sämtlicher Suchanfragen des Benutzers. Sie wird vom erweiterten Steuerprogramm eingerichtet und aktiviert, wenn der Benutzer die Dienste dieses speziellen KI-Assistenten in Anspruch nehmen möchte. Die Monitormaschine zerlegt den Informationsverarbeitungsprozess in zwei Zyklen:

Im ersten Zyklus wird für jede Suchanfrage, die der Benutzer bearbeiten möchte, eine eigene Maschine 'SUCHANFRAGE' generiert, die zunächst die Erfassungsphase durchläuft. Im zweiten Zyklus - wenn also die Erfassung und Korrektur sämtlicher Suchanfragen beendet ist - wählt der Benutzer nacheinander jeweils eine seiner Suchanfrage aus, für die er die Präzisierungsphase durchlaufen möchte.

Die Zerlegung des Gesamtprozesses in die beiden Zyklen erfolgt mit der Absicht, den ersten Zyklus offline auf dem PC des Benutzers ablaufen zu lassen, weil für diesen Zyklus noch keine Kommunikation mit dem IRS erforderlich ist.

5.3 Die Maschine 'SUCHANFRAGE'

Die abstrakte Maschine 'SUCHANFRAGE' enthält in ihrem Datenobjekt alle Daten, die zu einer Suchanfrage des Benutzers gespeichert werden müssen. Wir definieren zunächst die Struktur einer Suchanfrage und beschreiben dann, wie der Benutzer bei ihrer Erstellung unterstützt wird. Dabei werden wir sehen, dass die Maschine 'SUCHANFRAGE' zur Erfüllung ihrer Aufgabe ein speziell strukturiertes Wörterbuch benötigt.

5.3.1 Die Struktur der Suchanfrage

Durch eine heuristisch begründete Gliederung der Suchanfrage soll der Benutzer dazu angeleitet werden, seinen Informationsbedarf nach Prinzipien zu formulieren, die sich in der Praxis bewährt haben.

Die Suchanfrage gliedert sich in vier Ebenen: in die Ebenen der Begriffe, der Satzaussagen, der Segmentaussagen und der Dokumentenaussagen.

a) *Ebene der Begriffe*

Die Begriffe bilden die kleinsten semantischen Einheiten des Themas, zu dem ein Informationsbedarf des Benutzers besteht. Die erste Aufgabe bei der Selektion von Dokumenten besteht somit in der Prüfung, ob in einem Dokument von einem bestimmten Begriff die Rede ist. Dazu wird im Dokument nach solchen Wortformen und Kollokationen gesucht, deren Auftreten als Indiz dafür gewertet werden kann, dass im referierten Dokument von diesem

Begriff die Rede ist.[1] Zu diesem Zweck bietet der KI-Assistent dem Benutzer die Möglichkeit, einen Begriff zu benennen und dieser Benennung eine Liste von Wortformen und Kollokationen zuzuordnen. Eine solche Liste bezeichnen wir als ***Begriffskonstrukt*** und die Gesamtheit von Benennung und Begriffskonstrukt als ***Begriffsdeklaration***.

Jeder Begriffsdeklaration der Suchanfrage wird eine Referenznummer zugeordnet. In einem Fenster, das ständig auf dem Bildschirm zu sehen ist, werden die Benennungen der bereits deklarierten Begriffskonstrukte gemeinsam mit ihren Referenznummern angezeigt. Durch Angabe der Referenznummer kann der Benutzer in einem Begriffskonstrukt, das er neu formulieren möchte, auf ein bereits formuliertes Begriffskonstrukt Bezug nehmen. So können aus einfacheren Begriffskonstrukten schrittweise zunehmend kompliziertere aufgebaut werden.

Die Syntax einer Begriffsdeklaration wird durch die Grammatik G_B beschrieben.[2] Diese ist in Abbildung 12 dargestellt.

Ist ein <Truncation-Wort> angegeben, so wird im Dokument nach dem <Bezeichner> gesucht, wobei eine sich daran anschließende beliebige Symbolfolge unberücksichtigt bleibt. Durch <Restlänge> kann die maximale Länge dieser Symbolfolge angegeben werden. Die <Aufeinanderfolge> drückt aus, dass die zweite <Einfachalternative> unmittelbar auf die erste folgen muss. Die <Hintereinanderfolge> besagt, dass sich an die erste <Einfachalternative> maximal m Wörter anschließen können, ehe die zweite folgt. Dabei entspricht m der Anzahl der angeführten Punkte. Durch das <Nebeneinanderstehen> wird zum Ausdruck gebracht, dass die beiden <Einfachalternativen> in beliebiger Reihenfolge nebeneinander stehen. Das <Beieinanderstehen> besagt dagegen, dass sich zwischen die beiden <Einfachalternativen> maximal n Wörter schieben können, wobei die Reihenfolge der <Einfachalternativen> beliebig ist. Dabei repräsentiert n die Anzahl der angeführten Doppelpunkte.

[1] Vgl. beispielsweise Papka/Allan (1998).
[2] Die in der Grammatikbeschreibung verwendeten Zeichen und Symbole werden im Symbolverzeichnis (S. XV) erläutert.

Grammatik G_B:

< Begriffsdeklaration > ::= < Benennung >
" ' " < Begriffskonstrukt > " ' "

< Benennung > ::= < Bezeichner >

< Begriffskonstrukt > ::= < Alternative > $\left[\text{","} < \text{Alternative} >\right]_0^\infty$

< Alternative > ::= < Einfachalternative > | < Kollokation >

< Einfachalternative > ::= < Wortform > | < Referenz > |
< Truncation – Wort >

< Wortform > ::= < Bezeichner >

< Referenz > ::= < Zahl >

< Truncation – Wort > ::= < Bezeichner > "¤" < Restlänge >

< Restlänge > ::= < Zahl >

< Kollokation > ::= < Aufeinanderfolge > |
< Hintereinanderfolge > |
< Nebeneinanderstehen > |
< Beieinanderstehen >

< Aufeinanderfolge > ::= < Einfachalternative >
< Einfachalternative >

< Hintereinanderfolge > ::= < Einfachalternative > $\left[\text{"."}\right]_1^7$
< Einfachalternative >

< Nebeneinanderstehen > ::= < Einfachalternative > " / "
< Einfachalternative >

< Beieinanderstehen > ::= < Einfachalternative > $\left[\text{" : "}\right]_1^7$
< Einfachalternative >

Abbildung 12: Grammatik G_B für eine Begriffsdeklaration

b) Ebene der Satzaussagen

Während auf der Ebene der Begriffe die kleinsten semantischen Einheiten des Themas beschrieben werden, dient diese Ebene dazu, Begriffskonstrukte zu ***Satzaussagen*** zu verknüpfen. Dabei wird unter einer Satzaussage eine solche Aussage verstanden, die im Dokument innerhalb eines einzigen Satzes formuliert ist.[1]

c) Ebene der Segmentaussagen

Unter einer ***Segmentaussage*** verstehen wir eine Aussage, die über mehrere Sätze hinweg im Rahmen eines Segments des Dokuments anzutreffen ist.

d) Ebene der Dokumentenaussagen

Der Zweck dieser Ebene ist es, Aussagen zu beschreiben, die innerhalb des gesamten Dokuments zum Ausdruck kommen und die wir folglich als ***Dokumentenaussagen*** bezeichnen.[2]

Die Zuordnung einer Aussagendeklaration zur Ebene der Satz-, Segment- bzw. Dokumentenaussagen wird dem Benutzer durch das bereits erwähnte Bildschirm-Fenster sichtbar gemacht.

Für die Deklaration der Satzaussagen, der Segmentaussagen und der Dokumentenaussagen gilt die gemeinsame Grammatik G_A, die in der Abbildung 13 angegeben ist.

Die Grammatiken G_B und G_A werden durch die Kontextbedingung ergänzt, dass Referenzen nur in Form von Rückbezügen auf zuvor deklarierte Konstrukte gestattet sind. Durch diese Bedingung wird sichergestellt, dass die Formulierung der Suchanfrage keine Zyklen enthält.

Durch das beschriebene Vier-Ebenen-Modell sollen bei der Konstruktion der Suchanfrage die inhaltlichen Gesichtspunkte gegenüber den syntaktisch-strukturellen Erfordernissen stärker in den Vordergrund gerückt werden.

---

[1] Vgl. beispielsweise Murdock/Croft (2005).
[2] Vgl. beispielhaft Vechtomova/Robertson/Jones (2003).

Grammatik G_A :

< Aussagendeklaration > ::= < Benennung >
" ' " < Aussagenkonstrukt > " ' "

< Benennung > ::= < Bezeichner >

< Aussagenkonstrukt > ::= < Term > $\left[\text{"OR"} < \text{Term} >\right]_0^\infty$

< Term > ::= < Faktor > $\left[< \text{Operator} > < \text{Faktor} >\right]_0^\infty$

< Faktor > ::= < Referenz > |
"(" < Aussagenkonstrukt > ")"

< Referenz > ::= < Zahl >

< Operator > ::= "AND" | "NOT"

Abbildung 13: Grammatik G_A für eine Satz-, Segment- oder Dokumentenaussage

Der Benutzer wird veranlasst, zunehmend komplexere semantische Einheiten zu beschreiben. Er muss zunächst auf jeden Fall seine Begriffe beschreiben, die Verwendung der drei Aussagenebenen ist dagegen nicht obligatorisch. Der Benutzer kann beispielsweise Dokumentenaussagen unmittelbar aus Begriffskonstrukten aufbauen, ohne die Zwischenebenen der Satz- und Segmentaussagen verwenden zu müssen.

5.3.2 Die Bearbeitung der Suchanfrage

Die Maschine 'SUCHANFRAGE' tritt nach ihrer Generierung zunächst in die Erfassungsphase ein. Diese Phase beinhaltet die Erfassung, die Korrektur und die syntaktische Prüfung einer Suchanfrage des Benutzers. Dazu wird sofort mit der Generierung dieser Maschine das Teilsystem 'EDITIEREN' aufgebaut. Nach Beendigung der Erfassungsphase gibt die Maschine 'SUCHANFRAGE' die Steuerung wieder an die Maschine 'RECHERCHE' zurück. Erst zu dem Zeitpunkt, an dem der Benutzer die Suchanfrage präzisieren will, erhält die

Maschine 'SUCHANFRAGE' erneut die Steuerung und tritt dann in die Präzisierungsphase ein.

In der Präzisierungsphase nimmt die Maschine 'SUCHANFRAGE' die Kommunikation mit dem IRS auf. Sie führt Selektionsoperationen in der Datenbank durch, informiert den Benutzer darüber, wie viele Dokumente mit der aktuellen Fassung der Suchanfrage selektiert werden, und unterstützt ihn bei der schrittweisen Präzisierung der Suchanfrage. Der Benutzer weist dabei die Maschine an, welche Bearbeitungsschritte in welcher Reihenfolge auszuführen sind. Der KI-Assistent baut für jeden dieser Bearbeitungsschritte ein Teilsystem abstrakter Maschinen auf:

1. Das Teilsystem 'SUCHANFRAGE-AUSFÜHRUNG IN STAIRS' stellt die Selektionsergebnisse bereit, die STAIRS zu einem Begriffs- oder Aussagekonstrukt ermittelt.

2. Durch das Teilsystem 'ERGEBNISSE ZEIGEN' werden dem Benutzer die Dokumente eines Selektionsergebnisses dargeboten.

3. Das Teilsystem 'EDITIEREN' bietet dem Benutzer die Möglichkeit, seine Konstrukte zu verändern und neue hinzuzufügen.

4. Durch das Teilsystem 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' (beschrieben im Abschnitt 5.4) wird eine statistische Analyse der Selektionsergebnisse durchgeführt, auf Grund derer dem Benutzer Wortformen zur Präzisierung seiner Begriffskonstrukte angeboten werden.

5. Durch das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' (beschrieben im Abschnitt 5.5) wird der Benutzer dazu angeleitet, seine Begriffskonstrukte zur Gesamt-Suchanfrage zu verknüpfen.

Die drei Teilsysteme 'SUCHANFRAGE-AUSFÜHRUNG IN STAIRS', 'ERGEBNISSE ZEIGEN' und 'EDITIEREN' dienen hauptsächlich dazu, organisatorische Aufgaben zu lösen. Sie führen im Niveau der Mensch-Maschine-Kommunikation nur unwesentlich über das Kommando-Niveau von STAIRS hinaus und werden deshalb hier nicht näher beschrieben.

Das Niveau der Kommunikation des Benutzers mit dem IRS wird allein durch die beiden Teilsysteme 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' und 'SUCHANFRAGE KONSTRUIEREN' angehoben. In diesen Teilsystemen tritt der KI-Assistent in Kommunikation sowohl mit dem IRS als auch mit dem Benutzer. Während das IRS Dokumente unabhängig vom konkreten Informationsbedarf des Benutzers speichert, kommt dem KI-Assistenten die Aufgabe zu, die aus dem IRS bereitgestellten Informationen mit den Wertungen des Benutzers zu verbinden. Derartige Wertungen fordert der KI-Assistent zu verschiedenen Zeitpunkten ab: Sie müssen deshalb in Lernprozessen[1] kumuliert werden.

Durch den KI-Assistenten werden zwei Arten von Lernprozessen verwirklicht: ein ***nichtüberwachter Lernprozess*** und ein ***überwachter Lernprozess***.[2]

Das Teilsystem 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' realisiert den nichtüberwachten Lernprozess. Dieses Teilsystem extrahiert Wortformen aus denjenigen Dokumenten, die durch ein Konstrukt der Suchanfrage des Benutzers selektiert wurden. In einem Lernprozess erwirbt nun der KI-Assistent - ohne Belehrung durch den Benutzer - die Fähigkeit, diese Wortformen zu normieren, um dem Benutzer ausgewählte normierte Wortformen zur Präzisierung der Begriffskonstrukte anzubieten.

Das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' realisiert den überwachten Lernprozess. Dem Benutzer werden Dokumente zur Relevanzbewertung vorgelegt. Den bewerteten Dokumenten werden Wortformen entnommen und mit der Relevanzbewertung verknüpft.

Mit den beiden geschilderten Lernprozessen werden zwei Ziele verfolgt:

1. Die Fähigkeit des KI-Assistenten zur Wortnormierung und zum Offerieren von Wortformen soll ausgebaut werden.

2. Der KI-Assistent soll auf Grund der Analyse von Dokumenten einerseits und auf Grund der Auswertung ihrer Relevanzbewertung durch den Benutzer andererseits die Fähigkeit erwerben, Vorschläge zu unterbreiten, wie der Benutzer seine Begriffskonstrukte zu einer Suchanfrage verknüpfen kann.

---

[1] Vgl. beispielsweise Bishop (2006).
[2] Vgl. beispielhaft Russell/Norvig (2004), S. 793 ff.

Zur Kumulierung des „Wissens“ über Wortformen, das aus den beiden Lernprozessen „geschöpft“ wird, verwendet die Maschine 'SUCHANFRAGE' ein in besonderer Weise strukturiertes Wörterbuch, welches durch das Teilsystem 'WÖRTERBUCH' realisiert wird. In Vorbereitung auf die Beschreibung der beiden Lernprozesse befassen wir uns im folgenden Abschnitt zunächst mit dem Modell und der Struktur des Teilsystems 'WÖRTERBUCH'.

5.3.3 Das Teilsystem 'WÖRTERBUCH'

Das Teilsystem 'WÖRTERBUCH' soll im Rahmen des KI-Assistenten zwei Aufgaben erfüllen:

1. Es soll die Fähigkeit erwerben, dem Benutzer normierte Wortformen anzubieten, die dieser in seine Begriffskonstrukte aufnehmen kann.[1]

2. Es soll Wortformen derart verwalten, dass der KI-Assistent in den relevanzbewerteten Dokumenten dieselben Selektionsoperationen ausführen kann wie in der gesamten Datenbank. Die Notwendigkeit dieser Aufgabe wird bei der Beschreibung des überwachten Lernprozesses deutlich werden.

Zur Lösung der ersten Aufgabe ist das Wörterbuch als ein lernendes System zu gestalten. Sein „Wissen“ resultiert dabei aus drei Komponenten:

1. Die erste Komponente ist eine ***Endungsmenge*** für die Wortnormierung. Sie fungiert als „Vorerfahrung“, die schon zum Zeitpunkt des Einrichtens des Wörterbuchs vorhanden ist. Dieses „Wissen“ ist unabhängig vom konkreten Informationsbedarf eines Benutzers.

2. Die zweite Komponente liegt in Form von ***Regeln*** vor. Diese Regeln werden dazu verwendet, Wortformen zu normieren und jene Wortformen herauszufiltern, die mit hoher Wahrscheinlichkeit für eine Präzisierung der Begriffskonstrukte geeignet sind. Der benutzerspezifische Charakter dieses „Wissens“ ergibt sich daraus, dass der Benutzer die Regeln auswählt und/oder parametrisiert.

[1] Vgl. beispielsweise Witten/Nevill-Manning/Maulsby (1996).

3. Die dritte Komponente bilden ***bewertete Wortformen***. Dabei ist zwischen dem überwachten und dem nichtüberwachten Lernprozess zu unterscheiden.

   Im nichtüberwachten Lernprozess resultiert die Bewertung der Wortformen daraus, dass die Wortformen alle aus Dokumenten stammen, die durch ein Konstrukt der Suchanfrage des Benutzers selektiert wurden. Die vom Wörterbuch zu erlernende Fähigkeit, dem Benutzer Wortformen anzubieten, basiert auf der Strategie, die Häufigkeit des Auftretens einer Wortform im Selektionsergebnis mit der Häufigkeit ihres Auftretens in der gesamten Datenbank zu vergleichen.

   Im überwachten Lernprozess wird die Relevanzbewertung, die der Benutzer für ein konkretes Dokument vornimmt, auf alle in diesem Dokument enthaltenen Wortformen übertragen und im Wörterbuch kumuliert. Die zu erlernende Fähigkeit, dem Benutzer Wortformen anzubieten, basiert nun auf der Strategie, die Häufigkeit des Auftretens einer Wortform in den relevanten Dokumenten mit der Häufigkeit ihres Auftretens in den irrelevanten Dokumenten zu vergleichen.

   Der in das Wörterbuch einfließende Strom von Wortformen bewirkt auch einen Zuwachs von „Wissen“ über die Durchführung von Wortnormierungen. Die als „Vorerfahrung“ in das Wörterbuch eingebrachte Endungsmenge beschreibt nur die Möglichkeit, eine Symbolfolge am Ende einer Wortform als Endung zu betrachten. Diese wird jedoch erst dann wirklich als Endung behandelt, wenn ein Lernereignis eingetreten ist, durch das sie als tatsächliche Endung bestätigt wird.

5.3.3.1 Modell des Wörterbuchs

Wir wollen nun das Modell des Wörterbuchs beschreiben und beginnen mit der Definition einiger Begriffe.

Als ***Wortform*** bezeichnen wir eine Symbolfolge, die sowohl aus den Buchstaben des Alphabets $L = \{A, B, ..., Z, a, b, ..., z\}$ als auch aus den Ziffern des Alphabets $D = \{0,1,2,3,4,5,6,7,8,9,0\}$ bestehen kann. Sie soll stets mit einem Buchstaben beginnen, an den sich eine beliebige Folge von Buchstaben oder Ziffern anschließen kann. Entsprechend der Aufgabenstellung sollen durch das

Eliminieren von Endungen Wortnormierungen durchgeführt werden.[1] Die Fähigkeit, Endungen zu erkennen, soll das Wörterbuch in einem Lernprozess erwerben. Als „Vorerfahrung“ wird dem Wörterbuch eine Menge $E = \{e_1, e_2, ..., e_n\}$ von Endungen mitgegeben. Bei der Zusammenstellung der Endungsmenge E werden nur „elementare“ Endungen berücksichtigt, aus denen Endungsfolgen gebildet werden können. Eine solche Endungsfolge ist Element der Plushülle

$$E^+ = \bigcup_{n \geq 1} E^n \tag{31}$$

Das einfachste Vorgehen bestünde darin, jedes Suffix, das in E^+ vorkommt, als Endungsfolge zu betrachten. In der Praxis führt das jedoch mitunter dazu, dass Symbolfolgen zu Unrecht als Endungsfolgen behandelt werden. So ist beispielsweise das Element „ ***en*** “ in der Wortform „ Transistor***en*** “ als Endung zu betrachten, in der Wortform „ Zeich***en*** “ dagegen nicht. Um solche Irrtümer zu vermeiden, soll eine Symbolfolge am Ende einer Wortform als ***bekräftigte Endungsfolge*** angesehen werden, wenn

a) sie in E^+ enthalten ist und

b) ein Lernereignis eingetreten ist, das sie als Endungsfolge bekräftigt.

Wir werden an späterer Stelle angeben, welche Situationen als derartige Lernereignisse betrachtet werden.

Ist im Falle einer Wortform

$$w = se \text{ mit } s = xy,\ x \in L,\ y \in (L \cup D)^* \text{ und } e \in E^+ \tag{32}$$

die Symbolfolge e eine bekräftigte Endungsfolge, so bezeichnen wir s als die ***normierte Wortform*** von w.

[1] Vgl. beispielsweise Porter (1980).

Wir beschreiben das Wörterbuch als eine zeitabhängige Datenstruktur. Das Wörterbuch zum Zeitpunkt t ist ein Quintupel

$$\mathcal{W}^t = \left(W^t, A^t, U^t, B^t, Q^t\right) \tag{33}$$

Dabei bedeuten:

W^t	eine Menge von Paaren (w,β), in denen der Wortform w das kumulierte Bewertungsmaß β zugeordnet ist;
A^t	die Menge der Wortformen, die der Benutzer in seinen Begriffskonstrukten verwendet hat;
U^t	die Menge der Wortformen, die aus unbewerteten Dokumenten stammen;
B^t	die Menge der Wortformen, die aus relevanzbewerteten Dokumenten stammen;
Q^t	die Menge derjenigen Wortformen, die zwar in das Wörterbuch aufgenommen wurden, die aber dem Benutzer nicht angeboten werden sollen.

5.3.3.1.1 Aufbau des Wörterbuchs

Zum Zeitpunkt t_0, an dem das Wörterbuch eingerichtet wird, hat es die Form $\mathcal{W}^{t_0} = (\varnothing,\varnothing,\varnothing,\varnothing,\varnothing)$. Wird dem Wörterbuch zum Zeitpunkt t eine Wortform w mit einer Einzelbewertung b angeboten, so bewirkt das eine Veränderung des Wörterbuchs. Die Form der Einzelbewertung hängt von der Art des Lernprozesses ab, in dem die Wortform verarbeitet wird.

Im nichtüberwachten Lernprozess stammt die Wortform w aus einem unbewerteten Dokument. Das kumulierte Bewertungsmaß β soll in diesem Lernprozess angeben, wie oft w bisher im Selektionsergebnis aufgetreten ist. Deshalb wird w dem Wörterbuch mit der Einzelbewertung $b = 1$ angeboten.

Im überwachten Lernprozess stammt die Wortform w aus einem relevanzbewerteten Dokument. Das kumulierte Bewertungsmaß enthält mit $\beta = (\beta_r, \beta_i)$ die Häufigkeiten, mit denen w bisher in relevanten (β_r) bzw. in irrelevanten (β_i) Dokumenten aufgetreten ist. Stammt w aus einem relevanten Dokument, so wird w dem Wörterbuch mit der Einzelbewertung $b = (1,0)$ angeboten, stammt dagegen w aus einem irrelevanten Dokument, so wird es mit der Einzelbewertung $b = (0,1)$ angeboten.

Sowohl im nichtüberwachten als auch im überwachten Lernprozess werden dem Wörterbuch gegebenenfalls auch Wortformen angeboten, die bereits in den Begriffskonstrukten der Suchanfrage enthalten sind. Diese Wortformen sollen jedoch keinen Einfluss auf die Zählerstände der kumulierten Bewertungsmaße haben - deshalb werden sie dem Wörterbuch im nichtüberwachten Lernprozess mit der Bewertung $b = 0$ und im überwachten Lernprozess mit $b = (0,0)$ angeboten.

Wir beschreiben nun, welche Veränderungen im Wörterbuch $\mathcal{W}^{t-1}$ eintreten, wenn ihm zum Zeitpunkt t eine Wortform w mit dem Bewertungsmaß b angeboten wird. Dabei sind zwei Situationen zu unterscheiden:

1. Die Wortform w ist schon in Form eines Paares (w, β) in der Menge W^{t-1} enthalten: Als neues kumuliertes Bewertungsmaß wird $\sigma(\beta, b)$ verwendet. Dabei ist σ eine Funktion, die wir später angeben werden.

2. Die Wortform w ist noch nicht in der Menge W^{t-1} enthalten. Je nachdem, ob w der Suchanfrage, einem unbewerteten Dokument oder einem bewerteten Dokument entstammt, wird w in eine der Mengen A^{t-1}, U^{t-1} bzw. B^{t-1} aufgenommen. Dann wird das Paar (w, b) zur Menge W^{t-1} hinzugefügt. Im Anschluss daran wird geprüft, ob sich in der nunmehr aktualisierten Menge W^t eventuell Veränderungen durch das Eliminieren von Endungsfolgen ergeben.

Die Veränderungen, die in der Menge W^{t-1} eintreten, können also wie folgt angegeben werden:

$$W^t = \begin{cases} \left\{W^{t-1} - \{(w,\beta)\} \cup \{(w, s(\beta,b))\} \right. & (w,\beta) \in W^{t-1} \\ \mathrm{Var}\left(W^{t-1} \cup \{(w,b)\}\right) & \text{sonst} \end{cases} \tag{34}$$

Mit Hilfe der Funktion $\mathrm{Var}(W)$ wird geprüft, ob in der Menge W ein Lernereignis eingetreten ist, durch das eine bekräftigte Endungsfolge erkannt wird. Ein solches Lernereignis kann in zweierlei Form eintreten:

1. Die Menge W enthält sowohl ein Element (s, β_1) als auch ein Element (se_2, β_2) mit $e_2 \in E^+$. Die Tatsache, dass die Symbolfolge s sowohl mit als auch ohne eine Endungsfolge auftritt, wird als Bekräftigung dafür betrachtet, dass s als normierte Wortform behandelt werden kann. Das kumulierte Bewertungsmaß der endungsbehafteten Wortform wird bei der normierten Wortform berücksichtigt; es wird also β_1 durch $s(\beta_1, \beta_2)$ ersetzt.

   Entstammt die endungsbehaftete Symbolfolge se_2 weder der Suchanfrage noch einem bewerteten Dokument, so kann sie „vergessen“ werden. In diesem Fall wird se_2 aus U und das Element (se_2, β_2) aus W entfernt. Außerdem wird die endungsbehaftete Symbolfolge se_2 aus der Menge Q entfernt, wenn sie in ihr enthalten war.

   Entstammt die endungsbehaftete Symbolfolge se_2 jedoch der Suchanfrage oder einem bewerteten Dokument, so muss sie weiterhin „im Gedächtnis“ bleiben. Sie wird dann zwar nicht aus W gestrichen, wird jedoch zur Menge Q hinzugefügt. Endungsbehaftete Wortformen aus relevanzbewerteten Dokumenten bleiben damit im Wörterbuch erhalten, werden aber nicht dem Benutzer angeboten.

2. Die Menge W enthält sowohl ein Element (se_1, β_1) als auch ein Element (se_2, β_2) mit $e_1, e_2 \in E^+$. Die Tatsache, dass die Symbolfolge s sowohl mit der Endungsfolge e_1 als auch mit der Endungsfolge e_2 auftritt, wird als Bekräftigung dafür betrachtet, dass s als normierte Wortform behandelt werden kann. In die Menge W wird das Paar $(s, \mathcal{S}(\beta_1, \beta_2))$ als neues Element aufgenommen. Die kumulierten Bewertungsmaße der endungsbehafteten Wortformen werden also bei der normierten Wortform aufbewahrt. Die normierte Wortform s wird in die Mengen A, B bzw. U aufgenommen, wenn se_1 und/oder se_2 in A, B bzw. U enthalten sind.

   Entstammt die endungsbehaftete Wortform se_i in einem der beiden Elemente (se_i, β_i); $i \in \{1,2\}$ weder der Suchanfrage noch einem bewerteten Dokument, so kann sie „vergessen“ werden: Das Element se_i wird aus W (und natürlich auch aus U bzw. Q) entfernt.

   Entstammt die endungsbehaftete Symbolfolge se_i jedoch der Suchanfrage oder einem bewerteten Dokument, dann bleibt sie „im Gedächtnis“ erhalten, wird aber zu Q hinzugefügt, also zur Menge derjenigen Wortformen, die dem Benutzer nicht angeboten werden sollen.

Die Funktion $\mathrm{Var}(W)$ beschreibt die Veränderungen, die im Wörterbuch dann ausgelöst werden, wenn eines der beiden Lernereignisse L1 bzw. L2 eingetreten ist:

$$\text{L1:} \quad (s, \beta_1), (se_2, \beta_2) \in W \quad \text{mit} \quad e_2 \in E^+$$

$$\text{L2:} \quad (se_1, \beta_1), (se_2, \beta_2) \in W \quad \text{mit} \quad e_1, e_2 \in E^+$$

Für die Veränderungen $\mathrm{Var}(\mathrm{W})$ in Abhängigkeit von den Lernereignissen L1 bzw. L2 ergibt sich der folgende Ausdruck:

$$\mathrm{Var}(\mathrm{W}) = \begin{cases} \mathrm{W} - \{(s,\beta_1)\} \cup \{(s, \boldsymbol{s}(\beta_1,\beta_2))\} & \\ \quad - \{(se_2,\beta_2) \mid se_2 \notin A,\ se_2 \notin B\} & \text{L1} \\ \mathrm{W} \cup \{(s, \boldsymbol{s}(\beta_1,\beta_2))\} & \\ \quad - \{(se_i,\beta_i) \mid i \in \{1,2\},\ se_i \notin A,\ se_i \notin B\} & \text{L2} \\ \mathrm{W} & \text{sonst} \end{cases} \tag{35}$$

Wir geben nun an, in welcher Weise die Kumulierung der Bewertungen durch die Funktion $\boldsymbol{s}$ erfolgt. Im nichtüberwachten Lernprozess führt die Funktion eine einfache Aufsummierung der Häufigkeiten der Wortformen durch:

$$\boldsymbol{s}(\beta', \beta'') = \beta' + \beta'' \tag{36}$$

Im überwachten Lernprozess soll die Häufigkeitszählung differenzierter erfolgen und berücksichtigen, ob eine Wortform einem relevanten bzw. einem irrelevanten Dokument entstammt. Beim Eintritt in den überwachten Lernprozess werden die Zählerstände gelöscht: $\beta = (\beta_r, \beta_i) = (0,0)$. Die Funktion $\boldsymbol{s}$ hat dann die Form:

$$\begin{aligned} \boldsymbol{s}(\beta', \beta'') &= \boldsymbol{s}\left((\beta'_r, \beta'_i), (\beta''_r, \beta''_i)\right) \\ &= \left((\beta'_r + \beta''_r), (\beta'_i + \beta''_i)\right) \end{aligned} \tag{37}$$

Jedesmal dann, wenn sich das kumulierte Bewertungsmaß β einer Wortform w geändert hat und wenn w nicht in Q^t enthalten ist, muss getestet werden, ob w dem Benutzer vorgelegt werden soll, damit er entscheiden kann, ob er diese

Wortform eventuell in seine Begriffskonstrukte aufnehmen möchte. Dazu wird eine H_0-Hypothese geprüft, die feststellt, ob die Wortform w vom Thema des Benutzers statistisch unabhängig ist. Ist die H_0-Hypothese abzulehnen, wird w dem Benutzer zur Entscheidung vorgelegt und dann in die Menge Q^t aufgenommen, damit w dem Benutzer kein zweitesmal angeboten wird:

$$Q^t = \begin{cases} Q^{t-1} \cup \{w\} & w \notin Q^{t-1} \wedge H_0\text{ - Hypothese wird abgelehnt} \\ Q^{t-1} & \text{sonst} \end{cases} \tag{38}$$

Wie die H_0-Hypothese konkret formuliert wird und welche Daten bei ihrer Prüfung verwendet werden, hängt davon ab, ob die Prüfung im nichtüberwachten bzw. im überwachten Lernprozess erfolgt.

Im nichtüberwachten Lernprozess wird eine Wortform w mit $(w, \beta) \in W^t$ dann dem Benutzer zur Entscheidung vorgelegt, wenn ihre Zweipunktverteilung im Selektionsergebnis signifikant von ihrer Zweipunktverteilung in der gesamten Datenbank abweicht. Die H_0-Hypothese lautet somit:

> H_0: Das Ereignis, dass w in einem selektierten Dokument auftritt, ist unabhängig von dem Ereignis, dass w in einem Dokument der Datenbank auftritt.

Die H_0-Hypothese wird bei einer Irrtumswahrscheinlichkeit α durch einen Vierfeldertest[1] mit der folgenden Vierfeldertafel geprüft:

[1] Vgl. beispielsweise Sachs (1972), S. 269 ff.

	w tritt auf	w tritt nicht auf
Dokument liegt im Selektionsergebnis	$A = \beta$	$B = (n_g - \beta)$
Dokument liegt in der Datenbank	$C = N_w$	$D = (N_g - N_w)$

Dabei ist n_g die Gesamtanzahl der Wörter, die bisher den Dokumenten des Selektionsergebnisses entnommen wurden; sie wird beim Aufbau des Wörterbuchs mitgezählt. Die Anzahl N_w ist die Häufigkeit, mit der die Wortform w in der Datenbank auftritt. Dieser Wert lässt sich durch eine Kommunikation des KI-Assistenten mit der Maschine 'IRS' ermitteln. Der Wert N_g entspricht der Gesamtanzahl aller Wörter in der Datenbank. Dieser Wert kann mit Hilfe der Kommandosprache von STAIRS nicht abgefragt werden. Er wird deshalb durch den Ausdruck

$$N_g = N_D \cdot N_T \tag{39}$$

geschätzt. Dabei ist N_D die Anzahl der Dokumente in der Datenbank. Dieser Wert kann im BROWSE-Zyklus durch Kommunikation mit der Maschine 'IRS' ermittelt werden. N_T ist der Mittelwert der Anzahl der aus einem Dokument entnommenen Wortformen. Dieser Wert wird beim Aufbau der Datenbank festgestellt.

Im überwachten Lernprozess wird eine Wortform w mit $(w,(\beta_r, \beta_i)) \in W^t$ dann dem Benutzer zur Entscheidung angeboten, wenn ihre Zweipunktverteilung in den bisher als relevant bewerteten Dokumenten signifikant von ihrer Zweipunktverteilung in den bisher als irrelevant bewerteten Dokumenten abweicht.

Die H_0-Hypothese lautet in diesem Fall:

> H_0: Das Ereignis, dass w in einem relevanten Dokument auftritt, ist unabhängig von dem Ereignis, dass w in einem irrelevanten Dokument auftritt.

Die H_0-Hypothese wird auch in diesem Fall bei einer Irrtumswahrscheinlichkeit α durch einen Vierfeldertest geprüft. Die Vierfeldertafel hat nun folgende Gestalt:

	w tritt auf	w tritt nicht auf
Dokument ist relevant	$A = \beta_r$	$B = (n_r - \beta_r)$
Dokument ist irrelevant	$C = \beta_i$	$D = (n_i - \beta_i)$

Die Anzahlen n_r bzw. n_i der bisher aus relevanten bzw. irrelevanten Dokumenten entnommenen Wortformen werden beim Aufbau des Wörterbuchs mitgezählt.

Die Prüfung beider H_0-Hypothesen erfolgt auf der Grundlage des $\hat{\chi}^2$-Wertes, der eine Funktion der Häufigkeiten A, B, C und D ist. Von seltenen Grenzfällen abgesehen, hat die Funktion die Form:

$$\hat{\chi}^2 = \frac{(A + B + C + D)(AD - BC)^2}{(A + B)(C + D)(A + C)(B + D)} \tag{40}$$

Die H_0-Hypothese ist abzulehnen, wenn $\hat{\chi}^2 > q$ gilt. Das Quantil q ist der Funktionswert der χ^2-Verteilung mit einem Freiheitsgrad und der Irrtumswahrscheinlichkeit α.

5.3.3.1.2 Suche im Wörterbuch

Im überwachten Lernprozess enthält die Menge W^t alle Wortformen, die bis zum Zeitpunkt t aus relevanzbewerteten Dokumenten extrahiert wurden. Zur Durchführung von Selektionsoperationen in diesen Dokumenten ist es erforderlich, die in den Begriffskonstrukten verwendeten Wortformen im Wörterbuch aufzusuchen.

Die Suche nach einer Wortform w in der Menge W^t erfolgt durch die Suchfunktion

$$F(w, W^t) = \begin{cases} w & (w, \beta) \in W^t \wedge w \in B^t \\ \varepsilon & \text{sonst} \end{cases} \tag{41}$$

Die Wortform w wird also nur dann gefunden, wenn sie aus einem relevanzbewerteten Dokument in das Wörterbuch aufgenommen wurde.

Bei der Suche nach einem Truncation-Wort $w¤n$ muss die Suchfunktion $T(w¤n, W^t)$ alle Wortformen w' bereitstellen, die aus relevanzbewerteten Dokumenten in das Wörterbuch aufgenommen wurden, w als Präfix besitzen und ein Suffix mit der maximalen Länge n haben. Wurde für das Suffix keine Längenbegrenzung n angegeben, so entspricht das der Angabe $w¤\infty$. Die Funktion T hat die Form:

$$\begin{aligned} &T(w¤n, W^t) \\ &= \left\{ w' \mid (w', \beta) \in W^t \wedge w' \in B^t \wedge w' \in \{w\} \left(\bigcup_{i \le n} (L \cup D)^i \right) \right\} \end{aligned} \tag{42}$$

5.3.3.2 Entwurf des Teilsystems 'WÖRTERBUCH'

Wir wenden uns nun der Aufgabe zu, aus dem Modell des Wörterbuchs eine zweckmäßige Architektur für ein Teilsystem abstrakter Maschinen abzuleiten. Dieses Teilsystem ist als UML-Paketdiagramm in Abbildung 14 dargestellt.

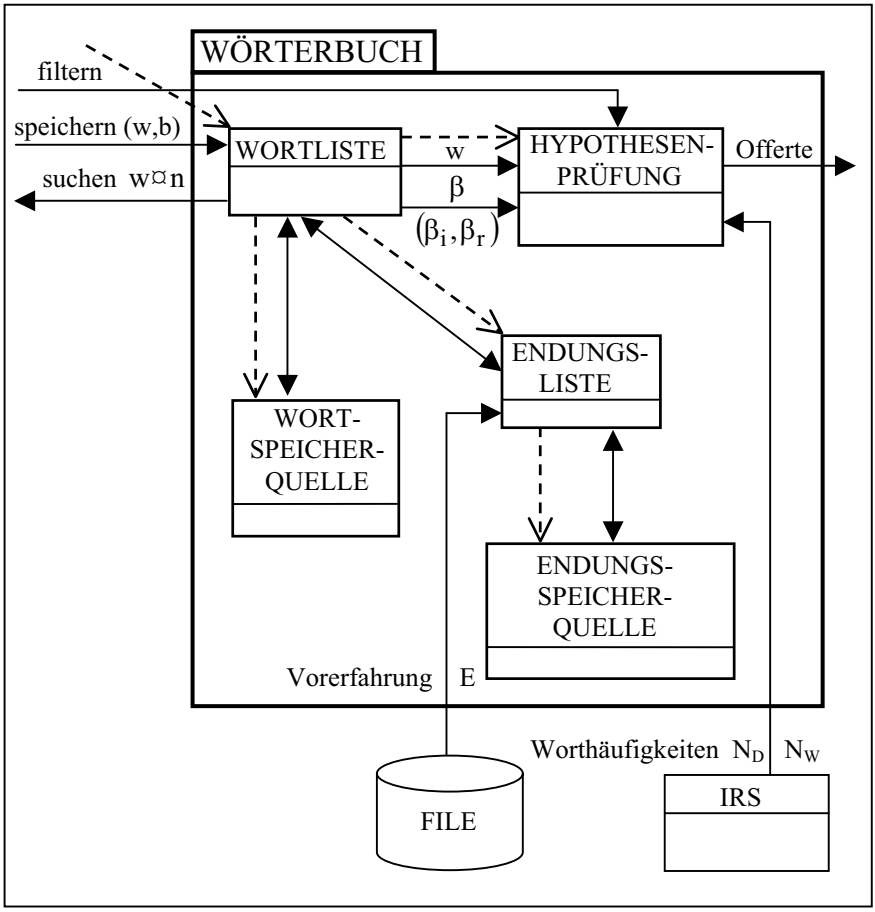


Abbildung 14: Struktur und Kommunikationsflüsse des Teilsystems 'WÖRTERBUCH'

Das Speichern von Paaren (w,b) und das Suchen von Wortformen w sowie von Truncation-Wörtern $w¤n$ (ausgefüllte Aufruf-Pfeile) erfolgt durch Aktivierung einer Maschine 'WORTLISTE', die als abstrakter Datentyp konzipiert wird. Daraus ergibt sich der Vorteil, dass alle Maschinen, die die Dienste der Maschine 'WORTLISTE' in Anspruch nehmen, unabhängig davon entworfen werden können, in welcher Weise diese ihre Aufgabe erfüllt.

Zur Verwaltung des Speicherbereichs für die Wortformen wird von der Maschine 'WORTLISTE' eine Maschine 'WORTSPEICHERQUELLE' generiert (gestrichelter Pfeil). Zur Verwaltung der „Vorerfahrung“ des Wörterbuchs wird eine Maschine 'ENDUNGSLISTE' eingerichtet, die wiederum eine Maschine 'ENDUNGSSPEICHERQUELLE' zur Verwaltung des Speicherbereichs für die Endungen generiert.

Eine eigenständige Aufgabe des Teilsystems 'WÖRTERBUCH' besteht darin, für eine Wortform w die H_0-Hypothese zu prüfen und die sich dabei qualifizierenden Wortformen als Offerten bereitzustellen. Zur Lösung dieser Aufgabe generiert 'WORTLISTE' eine Maschine 'HYPOTHESENPRÜFUNG'.

Man könnte nun jedes Mal, wenn sich in W^t das kumulierte Bewertungsmaß einer Wortform ändert, die Maschine 'HYPOTHESENPRÜFUNG' aktivieren. Da jedoch viele Wortformen nur sehr selten auftreten und es nicht sinnvoll ist, für seltene Ereignisse statistische Hypothesen zu prüfen, sollen die beiden Maschinen 'WORTLISTE' und 'HYPOTHESENPRÜFUNG' in anderer Weise zusammenarbeiten: Die Maschine 'WORTLISTE' gibt nicht jede Wortform, deren kumuliertes Bewertungsmaß sich geändert hat, an die Maschine 'HYPOTHESENPRÜFUNG' weiter, sondern nur jene Wortformen, die eine vorgegebene Häufigkeitsschranke H_{min} erreicht haben. Dieses Ereignis tritt für jede konkrete Wortform natürlich höchstens einmal auf. Mit der Übergabe einer Wortform w an die Maschine 'HYPOTHESENPRÜFUNG' erhält diese zugleich auch die Aufgabe, w während des gesamten weiteren Zeitverlaufs zu überwachen. 'HYPOTHESENPRÜFUNG' ist verpflichtet, die Wortform w auch dann weiter „im Auge zu behalten“, wenn sie sich nicht sofort statistisch qualifiziert hat.

Im nichtüberwachten Lernprozess kann sich die Zweipunktverteilung von w nämlich im Zuge der Analyse weiterer Dokumente des Selektionsergebnisses in der Weise verschieben, dass sie doch noch in einen signifikanten Kontrast zur Zweipunktverteilung von w in der Datenbank gerät.

Im überwachten Lernprozess kann es im Zuge der Relevanzbewertung weiterer Dokumente zu signifikanten Unterschieden zwischen den Zweipunktverteilungen von w in den relevanten bzw. in den irrelevanten Dokumenten kommen.

Alle Wortformen w, die 'WORTLISTE' an 'HYPOTHESENPRÜFUNG' übergibt, werden dort zwischengespeichert. Durch eine spezielle Aktivierung wird die Maschine 'HYPOTHESENPRÜFUNG' zu einem bestimmten Zeitpunkt veranlasst, für alle gespeicherten Wortformen die H_0-Hypothese zu prüfen und damit ein „Filtern“ der Wortformen vorzunehmen. Dafür benötigt die Maschine 'HYPOTHESENPRÜFUNG' Daten über die aktuellen Zweipunktverteilungen der Wortform in den analysierten Dokumenten, die sie sich durch Kommunikation mit der Maschine 'WORTLISTE' verschafft.

Im Verlaufe des nichtüberwachten Lernprozesses benötigt die Maschine 'HYPOTHESENPRÜFUNG' darüber hinaus noch die Zweipunktverteilung der Wortform in der Datenbank. Diese fordert sie von der Maschine 'IRS' ab. Das erfolgt für jede Wortform sinnvollerweise natürlich nur einmal: die Daten werden in 'HYPOTHESENPRÜFUNG' „gemerkt“.

Qualifiziert sich bei der Prüfung der H_0-Hypothese eine Wortform, so wird sie aus dem Zwischenspeicher entfernt und als Offerte bereitgestellt. Andernfalls verbleibt sie im Zwischenspeicher und wartet auf die nächste Gelegenheit, den statistischen Test zu bestehen.

Die Aufgaben, die die abstrakten Maschinen 'HYPOTHESENPRÜFUNG', 'WORTSPEICHERQUELLE' und 'ENDUNGSSPEICHERQUELLE' zu erfüllen haben, tragen vorwiegend organisatorischen Charakter. Deshalb wollen wir sie hier nicht näher behandeln. Wir konzentrieren uns auf die Beschreibung der Maschinen 'ENDUNGSLISTE' und 'WORTLISTE', die anspruchsvollere Aufgaben zu lösen haben. Zuvor klären wir jedoch, nach welcher Methode Wortmengen verwaltet werden.

5.3.3.2.1 Verwaltung einer Wortmenge durch einen Präfixbaum

Im Maschinensystem 'WÖRTERBUCH' müssen Wortmengen verwaltet werden. Die Wörter dieser Wortmengen müssen zu beliebigen Zeitpunkten gespeichert und wiederaufgefunden werden können. Beim Entwurf einer dafür geeigneten Datenstruktur sollen drei wesentliche Eigenschaften der Wörter Berücksichtigung finden:

1. Die Wörter weisen unterschiedliche Längen auf.

2. Es gibt Wörter, die in einem Präfix übereinstimmen.

3. Zu jedem Wort sind Attribute zu speichern.

Als Speicherstruktur für eine solche Wortmenge wählen wir eine rekursive Datenstruktur, die wir als ***Präfixbaum*** bezeichnen.

Ein Präfixbaum Ψ_C dient zur Verwaltung einer Klasse von Wörtern W_C, die eine gemeinsame Struktur C aufweisen. Die Struktur C wird durch zwei Eigenschaften bestimmt:

1. die Wörter stimmen in einem Präfix π überein und

2. das auf das Präfix π folgende Symbol z ist in einer vorgegebenen Menge von Symbolen $\{z_1, z_2, ..., z_i\}$ ***nicht*** enthalten.

Wir symbolisieren die Struktur C durch die Notation $C = \pi / \bar{z}_1 \bar{z}_2 ... \bar{z}_i /$. Der Präfixbaum $\Psi_C = \Psi_{\pi / \bar{z}_1 \bar{z}_2 ... \bar{z}_i /}$ ist zuständig für die Verwaltung der Wortklasse

$$W_{\pi / \bar{z}_1 \bar{z}_2 ... \bar{z}_i /} = \{ w \mid w = \pi z \rho \ \wedge \ z \notin \{z_1, z_2, ..., z_i\} \} \qquad (43)$$

Der Präfixbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$ muss vier Komponenten enthalten:

1. das Symbol z, das auf das Präfix π folgt, und das nicht Element der Menge $\{z_1, z_2, \ldots, z_i\}$ ist,

2. die Menge $V_{\pi z}$ der Attribute, die der Symbolfolge πz zugeordnet sind,

3. den Präfixbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/}$, der zur Verwaltung sämtlicher Wörter $w = \pi z'\rho$ mit $z' \notin \{z_1, z_2, \ldots, z_i, z\}$ dient, und

4. den Präfixbaum $\Psi_{\pi z//}$, der zur Verwaltung sämtlicher Wörter $w = \pi z z''\rho$ mit dem Präfix πz dient. Die Notation "//" bringt dabei zum Ausdruck, dass an das Symbol z'', das sich an das Präfix πz anschließt, keine einschränkenden Bedingungen gestellt werden.

Enthält eine Wortklasse W_C noch kein Element, so ist auch der entsprechende Präfixbaum Ψ_C leer. Ein Präfixbaum hat also die Form:

$$\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/} = \begin{cases} (z, V_{\pi z}, \Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/}, \Psi_{\pi z//}) & B \\ \varnothing & \text{sonst} \end{cases} \qquad (44)$$

Durch die Bedingung B wird die Forderung formuliert, dass die Wortmenge $W_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$ nicht leer ist:

$$B: \quad \exists(w)\left[w = \pi z\rho \text{ mit } z \notin \{z_1, z_2, \ldots, z_i\}\right]$$

Die gesamte gespeicherte Wortmenge wird durch den Präfixbaum $\Psi_{\varepsilon //}$ verwaltet, dessen rekursive Struktur in Abbildung 15 angedeutet ist.

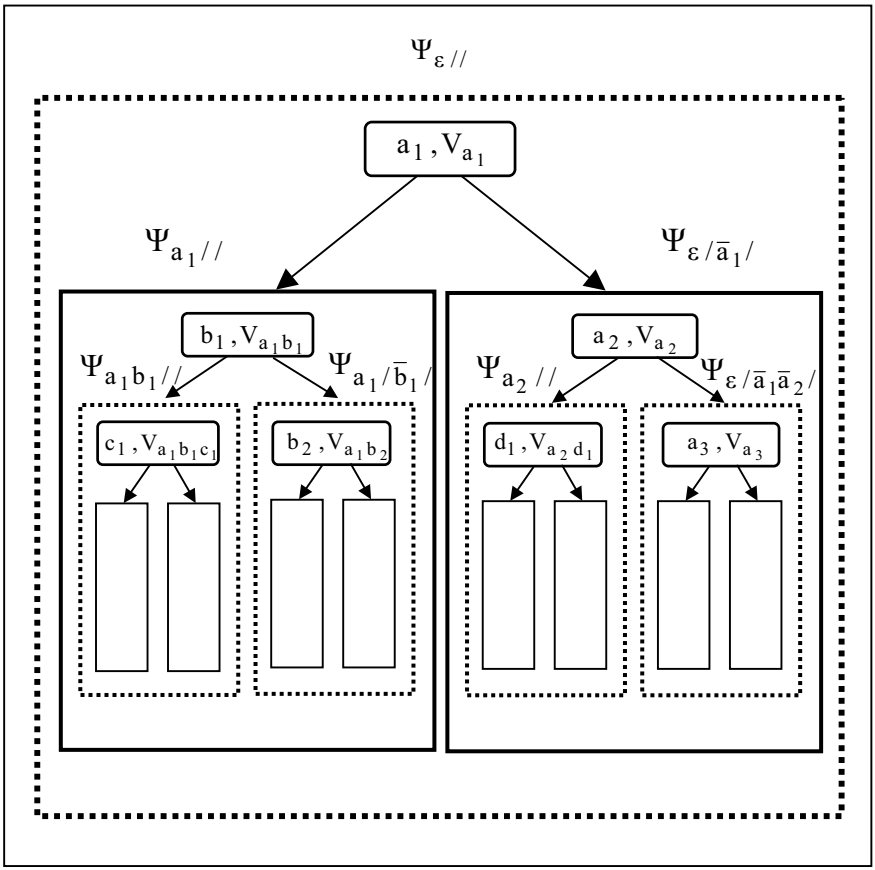


Abbildung 15: Rekursive Struktur des Präfixbaums

Wir veranschaulichen den Aufbau-Prozess eines Präfixbaums an einem einfachen Beispiel. In einen zunächst leeren Präfixbaum sollen nacheinander die Wörter „MIT“, „MANN“, „UND“, „MAUS“ eingespeichert werden. Als

Attribut zum Präfix π soll lediglich angegeben werden, ob π mit einem eingespeicherten Wort übereinstimmt.

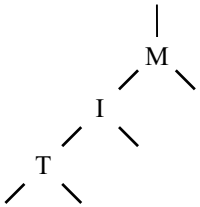
Neben der algebraischen Darstellung ist der entstandene Präfixbaum in einer vereinfachten graphischen Darstellung abgebildet.

Anfangssituation:

$\Psi_{\varepsilon //} = \varnothing$ (leerer Präfixbaum)

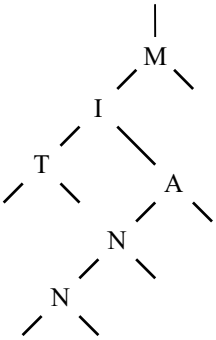
Nach Einspeichern des ersten Wortes „MIT“:

$$\Psi_{\varepsilon //} = (\mathrm{M}, \mathbf{false}, \varnothing, \Psi_{\mathrm{M}//})$$
$$\Psi_{\mathrm{M}//} = (\mathrm{I}, \mathbf{false}, \varnothing, \Psi_{\mathrm{MI}//})$$
$$\Psi_{\mathrm{MI}//} = (\mathrm{T}, \mathbf{true}, \varnothing, \varnothing)$$



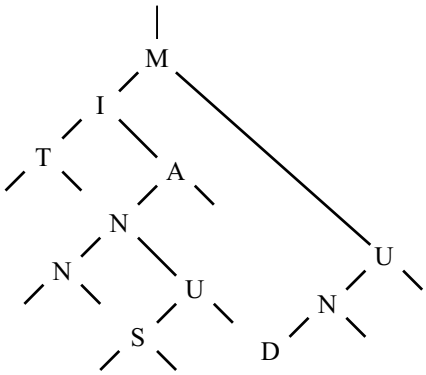
Nach Einspeichern des zweiten Wortes „MANN“:

$$\Psi_{\varepsilon //} = (\mathrm{M}, \mathbf{false}, \varnothing, \Psi_{\mathrm{M}//})$$
$$\Psi_{\mathrm{M}//} = (\mathrm{I}, \mathbf{false}, \Psi_{\mathrm{M}/\bar{\mathrm{I}}/}, \Psi_{\mathrm{MI}//})$$
$$\Psi_{\mathrm{M}/\bar{\mathrm{I}}/} = (\mathrm{A}, \mathbf{false}, \varnothing, \Psi_{\mathrm{MA}//})$$
$$\Psi_{\mathrm{MA}//} = (\mathrm{N}, \mathbf{false}, \varnothing, \Psi_{\mathrm{MAN}//})$$
$$\Psi_{\mathrm{MAN}//} = (\mathrm{N}, \mathbf{true}, \varnothing, \varnothing)$$
$$\Psi_{\mathrm{MI}//} = (\mathrm{T}, \mathbf{true}, \varnothing, \varnothing)$$



Schließlich nach Einspeichern des letzten Wortes „MAUS“:

Wir geben lediglich die graphische Darstellung an:



Die Funktion 'erweiterter Präfixbaum', durch die ein Wort w mit den Attributen v in den Präfixbaum $\Psi_{\varepsilon//}$ aufgenommen wird, ist rekursiv. Sie wird wie folgt aufgerufen:

$$\Psi_{\varepsilon//} := \text{erweiterter Präfixbaum}\left(w, v, \Psi_{\varepsilon//}\right);$$

Algorithmus: erweiterter Präfixbaum

Eingabe: Symbolfolge $c = c_1 c_2 \ldots c_n$
Attribute v
Präfixbaum $\Psi_{\pi/\bar{z}_1 \bar{z}_2 \ldots \bar{z}_i/}$

Ausgabe: Funktionswert Ψ

Vorgehen:

begin
if $\Psi_{\pi / \bar{z}_1 \bar{z}_2 \ldots \bar{z}_i /} = \varnothing$
then $\Psi_{\pi / \bar{z}_1 \bar{z}_2 \ldots \bar{z}_i /} :=$ neuer Präfixbaum (c, v)
else
begin
co Der Präfixbaum hat die Struktur:
$\Psi_{\pi / \bar{z}_1 \bar{z}_2 \ldots \bar{z}_i /} = \left(z, V_{\pi z}, \Psi_{\pi / \bar{z}_1 \bar{z}_2 \ldots \bar{z}_i \bar{z} /}, \Psi_{\pi z //}\right)$
oc
if $c_1 = z$ **then**
begin
co Für das Präfix πz gibt es einen Präfixbaum,
in dessen Wurzel wir gerade sind.
oc
if $c = c_1$ **then**
begin
co Das Wort πc_1 ist schon gespeichert **oc**
Übernehmen der Attribute v in $V_{\pi z}$;
end
else
begin
co Für das Wort $\pi z c_2 \ldots c_n$ ist der Präfixbaum
$\Psi_{\pi z //}$ zuständig.
oc
$\Psi_{\pi z //} :=$
erweiterter Präfixbaum $(c_2 \ldots c_n, v, \Psi_{\pi z //})$;
end;
end else

```
            begin
                co Für das Wort  πc1c2...cn  ist der Präfixbaum
                    Ψ π/z̄1z̄2...z̄iz̄/ zuständig
                oc
                Ψ π/z̄1z̄2...z̄iz̄/ :=
                erweiterter Präfixbaum (c, v, Ψ π/z̄1z̄2...z̄iz̄/ );
            end;
            Ψ := Ψ π/z̄1z̄2...z̄i/
        end;
    end
```

Durch die Funktion 'neuer Präfixbaum' wird für eine Symbolfolge ein Präfixbaum aufgebaut und als Funktionswert bereitgestellt:

Algorithmus: neuer Präfixbaum

Eingabe: Symbolfolge $c = c_1 c_2 \ldots c_n$
Attribute v

Ausgabe: Funktionswert Ψ

Vorgehen:

```
begin
    if  c = c1  then
    begin
        co Das Wortende ist erreicht oc
        Ψ := (c, v, ∅, ∅);
    end
    else  Ψ := (c1, ε, ∅, neuer Präfixbaum (c2...cn, v) );
end
```

Wie man eine Wortform bzw. ein Truncation-Wort in einem Präfixbaum aufsucht, wird durch den Algorithmus 'Wörter bereitstellen' beschrieben, der im Abschnitt 5.3.3.2.3 angegeben wird.

5.3.3.2.2 Die Maschine 'ENDUNGSLISTE'

Die Maschine 'ENDUNGSLISTE' ist ein integraler Bestandteil des Maschinensystems 'WÖRTERBUCH'. Sie wird mit der Lösung folgender Aufgaben betraut:

1. Aufbau eines Präfixbaums, der aus den Elementen der Endungsmenge E besteht, die in einer Datei gespeichert sind.

2. Prüfen, ob sich eine gegebene Symbolfolge ohne Rest in elementare Endungen zerlegen lässt.

Die zweite Aufgabe gehört zur gleichen Problemklasse wie die Zerlegung eines Wortes in seine Morpheme. Sie ist deshalb problematisch, weil die Endungsmenge E keinen präfixfreien (irreduziblen) Code bildet.[1] Sie erfüllt nämlich nicht die Fano-Bedingung, die eine Voraussetzung für die eindeutige Decodierbarkeit eines Codes mit variabler Wortlänge ist. Diese Bedingung lautet:

> ***Fano-Bedingung***:[2]
>
> Ein Code mit variabler Wortlänge muss so generiert werden, dass kein Code-Wort eines Zeichens mit dem Anfang des Code-Wortes irgendeines anderen Zeichens übereinstimmt.

In unserem Fall entspricht jedes zu codierende „Zeichen" einer Endung der Endungsmenge E. Ein „Code-Wort" ist jene Symbolfolge mit variabler Länge, durch die die Endung textmäßig repräsentiert wird. Die Fano-Bedingung ist deshalb nicht erfüllt, weil es ein Symbolfolge $e_1 \in E$ geben kann, die Präfix einer anderen Symbolfolge $e_2 \in E$ ist. Beginnt nun die zu analysierende Symbol-

[1] Vgl. beispielsweise Kämmerer (1974), S. 304 ff.
[2] Vgl. Ernst (2003), S. 70 ff.

folge mit e_2, so könnte als Präfix sowohl e_1 als auch e_2 abgespalten werden. Welche der beiden Möglichkeiten - und ob überhaupt eine von beiden - zu einer vollständigen Zerlegung der Symbolfolge führt, lässt sich erst entscheiden, wenn die Symbolfolge bis ans Ende analysiert wurde. Der Algorithmus muss deshalb ein „Backtracking“ vorsehen, um im Nachhinein zwischenzeitlich nicht verfolgte Varianten prüfen zu können.[1]

Beispielsweise werden durch die Wortpaare „ [der] Fisch und [die] Fisch***e*** “, „ [die] Zeit und [die] Zeit***en*** “ sowie „ [der] Tor und [die] Tor***heit*** “ die drei Endungen e_1 =„ ***e*** “, e_2 =„ ***en*** “ und e_3 =„ ***heit*** “ bestätigt, wobei e_1 Präfix von e_2 ist. Wenn nun geprüft wird, ob im Wortpaar „ [der] Verbund und [die] Verbund***enheit*** “ die Symbolfolge „ ***enheit*** “ als Endungsfolge bestätigt werden kann, könnte erst e_1 =„ ***e*** “ abgespalten werden - der Rest „ ***nheit*** “ ist dann keine Endungsfolge. Wird dagegen zuerst e_2 =„ ***en*** “ abgespalten, ist der Rest „ ***heit*** “ ebenfalls eine Endung.

Ob sich eine Symbolfolge w als eine Folge von Endungen der Menge E darstellen lässt, kann durch die rekursive Funktion Φ geprüft werden:

$$\Phi(w) = \begin{cases} \textbf{true} & w \in E \\ \textbf{true} & w = ew' \wedge e \in E \wedge \Phi(w') = \textbf{true} \\ \textbf{false} & \text{sonst} \end{cases} \tag{45}$$

Das Problem besteht also darin, von w diejenige Endung e abzuspalten, die einen Rest w' übriglässt, der aus einer Folge von Endungen besteht.

Die Speicherung der Endungen im Präfixbaum $\Psi_{\varepsilon//}$ erfolgt nach dem Algorithmus 'erweiterter Präfixbaum', der im vorigen Abschnitt angegeben wurde. Als Attribut eines Präfixes π wird die Angabe gespeichert, ob π mit einem Element $e \in E$ zusammenfällt. Die Prüfung, ob sich die Symbolfolge w ohne

[1] Vgl. beispielsweise Russell/Norvig (2004), S. 188 ff.

Rest in Endungen zerlegen lässt, erfolgt unter Verwendung des rekursiven Algorithmus' 'ist Endungsfolge', der wie folgt aufgerufen wird:

if ist Endungsfolge $\left(w, \Psi_{\varepsilon //}\right)$ **then** ...

Algorithmus: ist Endungsfolge

Eingabe: Symbolfolge $c = c_1 c_2 \ldots c_n$

Präfixbaum $\Psi_{\pi / \bar{z}_1 \bar{z}_2 \ldots \bar{z}_i /}$

Ausgabe: Funktionswert Φ

Vorgehen:

```
var zerlegbar;

begin
    if Ψ_{π/z̄1 z̄2 ... z̄i/} = ∅ then Φ := false else
    begin
        co Der Präfixbaum hat die Struktur:
           Ψ_{π/z̄1 z̄2 ... z̄i/} = (z, V_πz, Ψ_{π/z̄1 z̄2 ... z̄i z̄/}, Ψ_{πz//})
        oc
        if c1 = z then
        begin
            if V_πz then
            begin
                co Mögliche Trennstelle gefunden.
                   Ist der Rest in Endungen zerlegbar?
                oc
                if c = c1 then zerlegbar := true
                else zerlegbar: = ist Endungsfolge(c2 ... cn, Ψ_{ε//});
            end
            else zerlegbar := false;
```

```
                if zerlegbar then Φ := true else
                begin
                    co Keine gültige Trennstelle. Falls die Symbolfolge
                       noch nicht restlos analysiert wurde, muss weiter
                       gesucht werden.
                    oc
                    if c = c1
                    then Φ := false
                    else Φ := ist Endungsfolge( c2 ... cn, Ψ πz// );
                end;
            end else
            begin
                co Präfixbaum für πc1 noch nicht gefunden oc
                Φ := ist Endungsfolge( c1c2 ... cn, Ψ π/z̄1z̄2 ... z̄iz̄/ );
            end;
        end;
    end
```

5.3.3.2.3 Die Maschine 'WORTLISTE'

Durch die Maschine 'WORTLISTE' wird ein abstrakter Datentyp realisiert, der Wortformen mit ihren Attributen verwaltet. Über dem abstrakten Datentyp sind folgende Operationen erklärt:

1. Hinzufügen einer neuen Wortform,
2. Kumulieren der Attribute zu einer bereits gespeicherten Wortform,
3. Erkennen und Eliminieren von Endungsfolgen und
4. Aufsuchen aller Wortformen, die in einem vorgegebenen Präfix übereinstimmen.

Im Abschnitt 5.3.3.1 (Formel 33) wurde das Wörterbuch als ein Quintupel $\mathcal{W}^t = \left(W^t, A^t, U^t, B^t, Q^t\right)$ beschrieben. Die im Wörterbuch enthaltenen Wortformen werden einschließlich ihrer Attribute im Datenobjekt der Maschine 'WORTLISTE' in einem Präfixbaum verwaltet.

Zur Markierung des Beginns einer Wortform führen wir das Symbol "$\triangleleft$" ein. Jede Wortform w der Wortliste hat dann die Struktur:

$$w \in \{\triangleleft\}\, L(L \cup D)^* \tag{46}$$

Der Vorteil, den die Markierung des Wortbeginns bietet, wird später sichtbar werden. Zu einem Präfix π mit $w = \pi\rho\,;\ (w,\beta) \in W^t$ werden folgende Attribute gespeichert:

1. Das kumulierte Bewertungsmaß β_π des Präfixes π. Das ist die Summe der kumulierten Bewertungsmaße aller Wortformen aus W^t, die mit dem Präfix π beginnen. Im nichtüberwachten Lernprozess gilt:

$$\beta_\pi = \sum_{\substack{(w,\beta) \in W^t \\ w = \pi\rho}} \beta \tag{47}$$

und im überwachten Lernprozess:

$$\beta_\pi = \left(\sum_{\substack{(w,(\beta_r,\beta_i)) \in W^t \\ w = \pi\rho}} \beta_r \ , \sum_{\substack{(w,(\beta_r,\beta_i)) \in W^t \\ w = \pi\rho}} \beta_i \right) \tag{48}$$

Das kumulierte Bewertungsmaß β_π entspricht also im ersten Fall der Anzahl der Wortformen, die bisher aus dem Selektionsergebnis extrahiert wurden und π als Präfix haben. Im zweiten Fall enthält β_π die Häufigkeiten, mit denen π bisher in relevanten bzw. irrelevanten Dokumenten als Präfix einer Wortform vorkam.

Nun wird der Vorteil der Markierung des Wortbeginns durch das Symbol "$\triangleleft$" sichtbar: $\beta_\triangleleft$ ist nämlich das kumulierte Bewertungsmaß sämtlicher bereits gespeicherter Wortformen. Die in $\beta_\triangleleft$ enthaltenen Häufigkeiten werden gemäß (40) bei der Prüfung der H_0-Hypothese benötigt.

2. Das Attribut j_π, das angibt, ob das Präfix π mit einer in W^t gespeicherten Wortform übereinstimmt:

$$j_\pi = \begin{cases} \textbf{true} & \exists\left((w, \beta) \in W^t\right) \quad [w = \pi] \\ \textbf{false} & \text{sonst} \end{cases} \tag{49}$$

Dieses Attribut kennzeichnet jene Wortformen, die die Maschine 'WÖRTERBUCH' in der eingegebenen Form belassen hat oder die durch eine Eliminierung von Endungsfolgen gewonnen wurden.

3. Die Attribute a_π, u_π, b_π bzw. q_π, die angeben, ob π Präfix einer Wortform ist, die Element der Mengen A^t, U^t, B^t bzw. Q^t ist. Sie bringen also entweder zum Ausdruck, woher das Präfix stammt (a_π - aus einem Begriffskonstrukt, u_π - aus einem unbewerteten Dokument bzw. b_π - aus einem bewerteten Dokument) oder geben an, ob die mit ihm übereinstimmende Wortform dem Benutzer nicht als Offerte angezeigt werden soll (q_π).

Nachdem wir die Struktur des Datenobjekts der Maschine 'WORTLISTE' beschrieben haben, gehen wir auf die Operationen ein, die über dem abstrakten Datentyp erklärt sind.

Die Operationen des Hinzufügens einer neuen Wortform und des Kumulierens der Attribute zu einer bereits gespeicherten Wortform laufen ab, wie es bereits im Abschnitt 5.3.3.2.1 beschrieben wurde.

Tritt dabei jedoch die Situation ein, dass der Speicherbereich, der von der Maschine 'WORTLISTENQUELLE' verwaltet wird, erschöpft ist, so muss eine Speicherrückgewinnung eingeleitet werden. Speicherbereiche können nur dadurch freigesetzt werden, dass belegte Speicherbereiche an die Maschine 'WORTLISTENQUELLE' zurückgegeben werden. Da Wortformen aus der Suchanfrage des Benutzers oder aus relevanzbewerteten Dokumenten nicht „vergessen“ werden dürfen, kann die Speicherrückgewinnung nur auf Kosten jener Wortformen erfolgen, die ausschließlich im nichtüberwachten Lernprozess aufgetreten sind. Dabei wird das Ziel verfolgt, möglichst wenig „Wissen“ aufzugeben: Die Speicherrückgewinnung muss so erfolgen, dass die Fähigkeit der Maschine 'WORTLISTE' zum Eliminieren von Endungen und zum Herausfiltern von Offerten möglichst wenig geschmälert wird. Die Maschine 'WORTLISTE' berücksichtigt diese Forderung, indem sie die am seltensten aufgetretenen Wortformen zuerst „vergisst“. Nur dann, wenn der dabei freigesetzte Speicherbereich den Bedarf noch immer nicht deckt, wird die Speicherrückgewinnung auf dem nächsthöheren Häufigkeitsniveau wiederholt.

Die Speicherrückgewinnung erfolgt durch die ***Reduktion*** eines Präfixbaums. Ein Präfixbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/} = \left(z, V_{\pi z}, \Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/}, \Psi_{\pi z//}\right)$ wird bei einem vorgegebenen Häufigkeitswert H dann reduziert, wenn die folgenden vier Bedingungen erfüllt sind:

1. Das Präfix πz darf keine Wortform sein, die aus der Suchanfrage des Benutzers oder aus einem relevanzbewerteten Dokument stammt. Es muss also gelten: $a_{\pi z} = b_{\pi z} =$ **false**.

2. Das Attribut $q_{\pi z}$ muss den Wert **false** haben: πz darf also nicht zu jenen Wortformen gehören, für die die Kenntnis aufgehoben werden muss, dass sie dem Benutzer ***nicht*** angeboten werden sollen.

3. Das kumulierte Bewertungsmaß $\beta_{\pi z}$ muss mit dem vorgegebenen Häufigkeitswert H übereinstimmen.

4. Der Präfixbaum $\Psi_{\pi z//}$ muss leer sein: Es darf also keine Wortform der Struktur $\pi z \rho;\ \rho \neq \varepsilon$ geben.

Im Zuge der Reduktion des Präfixbaums $\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i/}$ laufen die folgenden Aktivitäten ab:

1. Der durch die Variablen z und $V_{\pi z}$ belegte Speicher wird freigegeben.

2. Die Verweise auf die beiden Teilbäume $\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i\bar{z}/}$ und $\Psi_{\pi z//}$ werden gelöscht.

3. Der Verweis auf den Präfixbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i/}$ wird durch den Verweis auf den Teilbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i\bar{z}/}$ ersetzt.

Dieses Vorgehen wird in Abbildung 16 veranschaulicht.

Der bei der Speicherrückgewinnung entstehende Präfixbaum wird durch den rekursiven Algorithmus 'reduzierter Präfixbaum' als Funktionswert bereitgestellt. Der reduzierte gesamte Präfixbaum wird durch den folgenden Aufruf der Funktion gewonnen:

$\Psi_{\varepsilon//}$:= reduzierter Präfixbaum ($\Psi_{\varepsilon//}$, H);

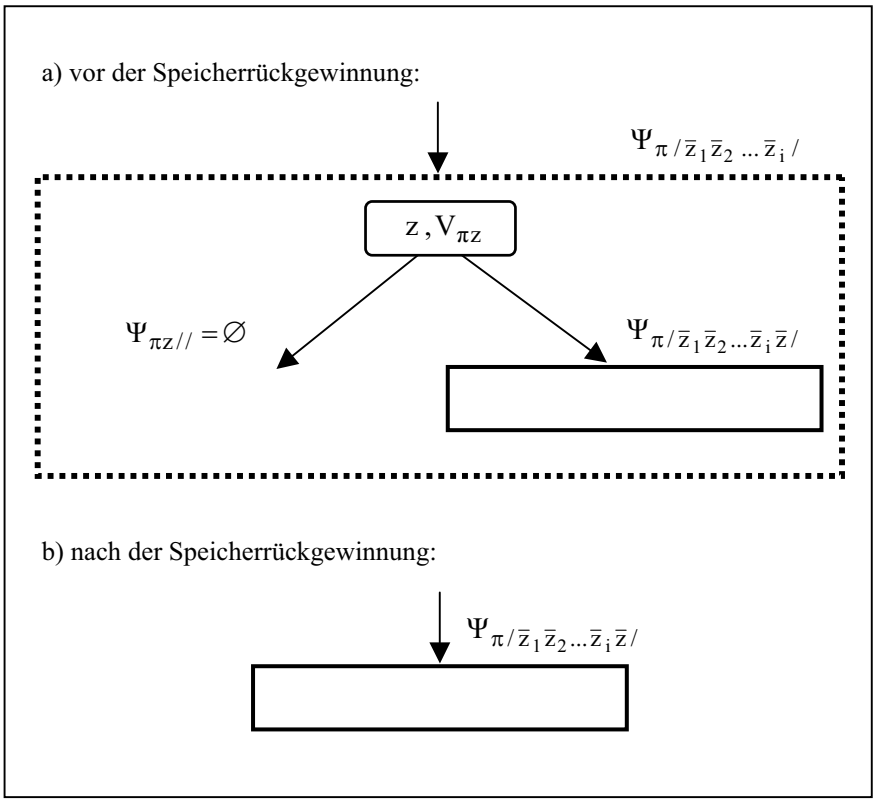


Abbildung 16: Präfixbaum vor (a) und nach (b) der Speicherrückgewinnung

Algorithmus: reduzierter Präfixbaum

Eingabe: Präfixbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$
Häufigkeitswert H

Ausgabe: Funktionswert Ψ

Vorgehen:

begin
if $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/} = \varnothing$ **then** $\Psi := \varnothing$ **else**
begin
co Der Präfixbaum hat die Struktur:
$\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/} = \left(z, V_{\pi z}, \Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/}, \Psi_{\pi z//}\right)$
oc
$\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/} :=$
reduzierter Präfixbaum ($\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/}$, H);
$\Psi_{\pi z//} :=$ reduzierter Präfixbaum ($\Psi_{\pi z//}$, H);
if $\Psi_{\pi z//} = \varnothing$ **then**
begin
co Es gibt keine Wortform der Struktur $\pi z\rho;\ \rho \neq \varepsilon$.
Wenn das Präfix πz nicht benötigt wird, kann
der Präfixbaum $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$ gelöscht werden.
oc
if Löschbedingung erfüllt ($V_{\pi z}$, H) **then**
begin
Speicherrückgabe von $\Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$;
$\Psi := \Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i\bar{z}/}$;
end else $\Psi := \Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$;
end else $\Psi := \Psi_{\pi/\bar{z}_1\bar{z}_2\ldots\bar{z}_i/}$;
end
end

Der erste Speicherrückgewinnungslauf erfolgt mit $H = 1$. Reicht der dadurch freigesetzte Speicherbereich noch nicht aus, so wird H schrittweise solange erhöht, bis ein Speicherbereich ausreichender Größe zurückgewonnen wurde.

Beim Erkennen und Eliminieren von Endungsfolgen in der Wortliste besteht das Hauptproblem in der Entscheidung, ob sich ein Suffix einer Wortform ohne Rest als Folge von elementaren Endungen darstellen lässt. Diese Aufgabe wird durch die bereits beschriebene Maschine 'ENDUNGSLISTE' gelöst. Es bleibt noch zu erläutern, in welcher Weise Wortformen mit übereinstimmendem Präfix in der Wortliste aufgesucht werden.

Im überwachten Lernprozess müssen die Wörter, die als <Wortform> oder als <Truncation-Wort> in den Begriffskonstrukten angegeben wurden, in der Wortliste aufgesucht werden. Beide Schreibweisen lassen sich auf die Form $w¤n$ zurückführen. Dabei ist w eine Symbolfolge, an die sich

a) kein weiteres Zeichen anschließen darf (im Fall $n = 0$) oder

b) maximal n beliebige Zeichen anschließen dürfen (im Fall $1 \le n < \infty$) oder aber

c) unbegrenzt viele beliebige Zeichen anschließen dürfen (im Fall $n \to \infty$).

Der Algorithmus 'Wörter bereitstellen' ermittelt alle in der Wortliste gespeicherten Wortformen, die der Bedingung $w¤n$ genügen. Er arbeitet als Prozess und stellt bei jeder Aktivierung die jeweils nächste Wortform bereit. Gibt es keine weitere solche Wortform, stellt der Algorithmus die Systemvariable `EOF` (***E**nd **O**f **F**ile*) auf den Wert **true**:

Algorithmus: Wörter bereitstellen

Eingabe: Bedingung $w¤n$

Ausgabe: Alle Wörter, die der Bedingung $w¤n$ genügen

Vorgehen:

co Vereinbarung der internen Algorithmen 'Wortformenbaum' und 'Wortformen bereitstellen', die aus Gründen der Übersichtlichkeit an späterer Stelle angegeben werden.
oc

var $\Psi_{w//}$

begin
co Zuerst wird im gesamten Präfixbaum $\Psi_{\varepsilon//}$ nach der Symbolfolge w gesucht **oc**
$\Psi_{w//} := \text{Wortformenbaum}(w, \Psi_{\varepsilon//})$;
if $\Psi_{w//} \neq \emptyset$ **then**
begin
co $\Psi_{w//}$ enthält die Wortform w sowie alle Wortformen der Struktur $w\rho$; $\rho \neq \varepsilon$. Diese werden bereitgestellt. **oc**
Wortformen bereitstellen ($\Psi_{w//}$, n);
end;
co Der Endpunkt ist erreicht **oc**
EOF := **true**; $s^+\left(\varepsilon, \underline{\text{aret}^+}\right)$;
end

Durch den rekursiven Algorithmus 'Wortformenbaum' wird w gesucht und - falls vorhanden - bereitgestellt. Der als Funktionswert übergebene Präfixbaum $\Psi_{w//}$ enthält alle Wortformen, die w als Präfix besitzen.

Algorithmus: Wortformenbaum

Eingabe: Symbolfolge $c = c_1 c_2 \ldots c_n$
Präfixbaum $\Psi_{\pi / \bar{z}_1 \bar{z}_2 \ldots \bar{z}_i /}$

Ausgabe: Funktionswert Ψ

Vorgehen:

begin
if $\Psi_{\pi / \overline{z}_1 \overline{z}_2 \ldots \overline{z}_i /} = \varnothing$ **then** $\Psi := \varnothing$ **else**
begin
co Der Präfixbaum hat die Struktur:
$\Psi_{\pi / \overline{z}_1 \overline{z}_2 \ldots \overline{z}_i /} = \left(z, V_{\pi z}, \Psi_{\pi / \overline{z}_1 \overline{z}_2 \ldots \overline{z}_i \overline{z} /}, \Psi_{\pi z //}\right)$
oc
if $c_1 = z$ **then**
begin
if $c = c_1$ **then**
begin
co Die Symbolfolge $w = \pi c_1$ ist gefunden **oc**
EOF := **false**; $s_{ar}(\pi c_1, \underline{aret})$; $\Psi := \Psi_{\pi z //}$;
end
else $\Psi :=$ Wortformenbaum$(c_2 \ldots c_n, \Psi_{\pi z //})$;
end
else $\Psi :=$ Wortformenbaum$(c_1 c_2 \ldots c_n, \Psi_{\pi / \overline{z}_1 \overline{z}_2 \ldots \overline{z}_i \overline{z} /})$;
end;
end

Der rekursive Algorithmus 'Wortformen bereitstellen' durchläuft den Präfixbaum $\Psi_{w //}$ und stellt in einem Aktivator-Rückkehrpunkt die jeweils nächste Wortform bereit:

Algorithmus: Wortformen bereitstellen

Eingabe: Präfixbaum $\Psi_{\pi / \overline{z}_1 \overline{z}_2 \ldots \overline{z}_i /}$
Suchtiefe t

Ausgabe: Sukzessive alle Wörter, die in $\Psi_{\pi / \overline{z}_1 \overline{z}_2 \ldots \overline{z}_i /}$ liegen und die Form $\pi\rho$; $\text{length}(\rho) \leq t$ haben

Vorgehen:

begin
if $\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i/} \neq \varnothing$ **and** $t > 0$ **then**
begin
co Der Präfixbaum hat die Struktur:
$\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i/} = \left(z, V_{\pi z}, \Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i\bar{z}/}, \Psi_{\pi z//}\right)$
oc
if $j_{\pi z}$ **then**
begin
co Die Symbolfolge πz ist eine gespeicherte Wortform, die der Bedingung $w \text{¤} n$ genügt
oc
EOF := **false**; $s_{ar}(\pi z, \underline{\text{aret}})$;
end;
Wortformen bereitstellen ($\Psi_{\pi/\bar{z}_1\bar{z}_2 \ldots \bar{z}_i\bar{z}/}, t$);
Wortformen bereitstellen ($\Psi_{\pi z//}, t-1$);
end;
end

5.3.3.3 Die durch das Teilsystem 'WÖRTERBUCH' realisierte Intelligenz

Das Teilsystem 'WÖRTERBUCH' erwirbt die Fähigkeit, Wortformen zu normieren und dem Benutzer ausgewählte Wortformen zur Präzisierung seiner Begriffskonstrukte anzubieten. In dieser Fähigkeit äußert sich seine Intelligenz.

In den Lernprozess des Teilsystems 'WÖRTERBUCH' wird „Vorerfahrung“ in Form einer Endungsliste eingebracht. Die „Vorerfahrung“ wird als unspezifisches Wissen behandelt, das weder die konkreten Informationen der Datenbank noch die Eingaben des Benutzers widerspiegelt. Sie ist also nicht auf den Informationsbedarf des Benutzers ausgerichtet. Sie spielt die Rolle einer notwendigen Bedingung für das Eliminieren einer Endungsfolge. Die hinreichende Bedingung dafür ist aber erst dann erfüllt, wenn ein Lernereignis eingetreten ist, durch das die unspezifische „Vorerfahrung“ bekräftigt wird.

Das Teilsystem 'WÖRTERBUCH' erfüllt die Funktion eines Filters. Mit der Verwaltung der Wortformen werden zugleich auch Informationen kumuliert, die eine problemspezifische Beurteilung der Wortformen zulassen. Die Beurteilung der Wortformen basiert auf statistischen Maßzahlen, deren Berechnung aufwendig ist. Um diesen Aufwand dann nicht treiben zu müssen, wenn seine Nutzlosigkeit von vornherein feststeht, erfolgt die Filterung in zwei Stufen: In der ersten Stufe wird mit geringem Aufwand für alle Wortformen eine notwendige Bedingung der Filterung geprüft. In der zweiten Stufe wird der hohe Aufwand zur Prüfung der hinreichenden Bedingung nur noch für diejenigen Wortformen getrieben, die sich in der ersten Stufe qualifiziert haben. Die Strategie des möglichst frühzeitigen Ausblendens unwesentlicher Informationen ist ein Grundprinzip der organismischen Informationsverarbeitung[1] und der automatisierten Massendatenverarbeitung.[2]

Zwischen dem Benutzer und dem Teilsystem 'WÖRTERBUCH' findet eine Rollenteilung statt, die für die Mensch-Maschine-Kommunikation typisch ist: Der Benutzer bewertet Dokumente hinsichtlich ihrer Relevanz für sein Thema. Mit Hilfe statistischer Verfahren filtert das Teilsystem 'WÖRTERBUCH' aus den Dokumenten Wortformen heraus. Über diese Wortformen entscheidet der Benutzer aus semantischer Sicht. Solche Systeme künstlicher Intelligenz, die dem Benutzer Offerten unterbreiten, enthalten im allgemeinen eine Erklärungskomponente, die dem Benutzer erläutert, auf Grund welcher Regeln die Offerte ausgewählt wurde. Das Wörterbuch verwendet jedoch als einzige Regel die statistische Hypothesenprüfung, so dass in diesem Fall auf die Erklärungskomponente verzichtet werden konnte.

5.4 Das Teilsystem 'BEGRIFFSKONSTRUKTE PRÄZISIEREN'

Der Benutzer hat in der Erfassungsphase der Suchanfrage seinen Informationsbedarf ohne Kommunikation mit dem Information-Retrieval-System STAIRS formuliert. Die deklarierten Begriffskonstrukte enthalten die ihm geläufigen Fachwörter zum Thema. Dabei ist es aber ungewiss, ob seine Formulierungen der Begriffs- und Aussagenkonstrukte bei einer Recherche in der Datenbank zu den gewünschten Ergebnissen führen.

---

[1] Vgl. Klix (1971).
[2] Vgl. Jarosch/Müller (1979).

Die Erfahrungen zeigen, dass bei der Konstruktion von Suchanfragen zu berücksichtigen ist, in welcher speziellen Weise in der jeweiligen Datenbank Begriffe und Aussagen dargestellt werden.[1] In der üblichen Praxis der Online-Recherche wird dieses Problem dadurch gelöst, dass sich der Benutzer Selektionsergebnisse am Bildschirm ansieht, den Dokumenten geeignete Wörter entnimmt und diese in seine Suchanfrage „einbaut". Bei diesem Vorgehen soll er nun vom KI-Assistenten unterstützt werden.

Das Teilsystem 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' befreit den Benutzer von der Durchsicht seines Selektionsergebnisses. Es analysiert die selektierten Dokumente und filtert aus ihnen jene Wortformen heraus, die mit großer Wahrscheinlichkeit zur Präzisierung der Begriffskonstrukte beitragen können. Der dabei ablaufende Informationsverarbeitungsprozess ist seinem Wesen nach ein nichtüberwachter Lernprozess. Das benutzerspezifische „Wissen" liegt im Selektionsergebnis und wird durch Kontrastbildung zur gesamten Datenbank nutzbar gemacht.

Das Teilsystem 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' löst die folgenden Teilaufgaben:

1. Die Dokumente des Selektionsergebnisses werden in Wortformen zerlegt.

2. Die Wortformen werden gesammelt. Normierte Wortformen, die für das Thema wesentlich sind, werden herausgefiltert.

3. Die herausgefilterten Wortformen werden dem Benutzer zur Entscheidung vorgelegt, ob er sie zur Präzisierung seiner Begriffskonstrukte verwenden möchte.

4. Der Benutzer wird beim Einfügen der ausgewählten Wortformen in seine Begriffskonstrukte unterstützt.

Dieser Zerlegung der Gesamtaufgabe in Teilaufgaben entspricht die Struktur des Maschinensystems 'BEGRIFFSKONSTRUKTE PRÄZISIEREN', das in der Abbildung 17 dargestellt ist.

[1] Vgl. beispielsweise Lewandowski (2005) und Poetzsch (2005).

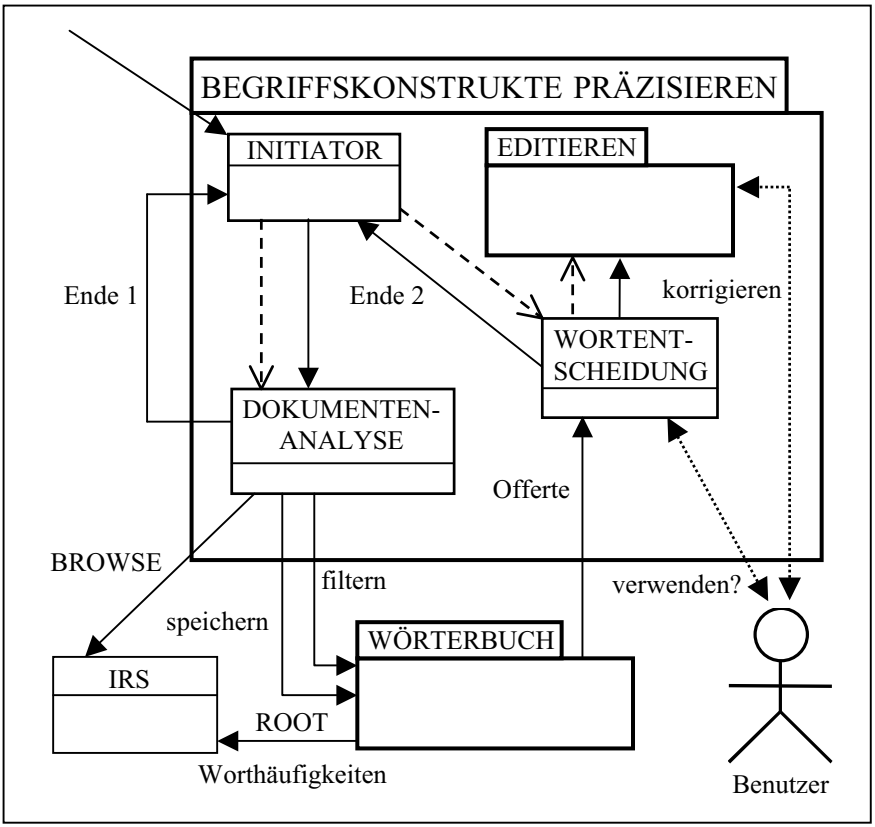


Abbildung 17: Struktur und Kommunikationsflüsse des Teilsystems 'BEGRIFFSKONSTRUKTE PRÄZISIEREN'

Das Teilsystem 'BEGRIFSSKONSTRUKTE PRÄZISIEREN' wird von der Maschine 'SUCHANFRAGE' durch den Aufruf der Maschine 'INITIATOR' aktiviert, die die Regie über den Ablauf des weiteren Informationsverarbeitungsprozesses übernimmt. Zur Lösung der Aufgabe, die Dokumente des Selektionsergebnisses in Wortformen zu zerlegen, generiert der 'INITIATOR' die Maschine 'DOKUMENTENANALYSE'. Der 'INITIATOR' generiert außerdem die Maschine 'WORTENTSCHEIDUNG', durch die eine Kommuni-

kation mit dem Benutzer zur Entscheidung über die offerierten Wortformen durchgeführt werden soll.

Mit der Aktivierung der Maschine 'DOKUMENTENANALYSE' beginnt der Informationsverarbeitungsprozess zur Ermittlung von Offerten. Dazu fordert 'DOKUMENTENANALYSE' in einem BROWSE-Zyklus jeweils ein Dokument von der Maschine 'IRS' ab und zerlegt es in Wortformen. Es ist die Aufgabe des Teilsystems 'WÖRTERBUCH', diese Wortformen zu speichern und dabei Wortnormierungen vorzunehmen. Sobald ein Dokument vollständig verarbeitet wurde, fordert die Maschine 'DOKUMENTENANALYSE' das Teilsystem 'WÖRTERBUCH' dazu auf, die wesentlichen Wortformen herauszufiltern. Zu diesem Zweck fordert 'WÖRTERBUCH' von der Maschine 'IRS' durch ROOT-Kommandos Informationen über die Häufigkeitsverteilung der normierten Wortformen in der Datenbank ab. Alle Wortformen, die auf Grund der statistischen Analyse als wesentlich erkannt wurden, werden vom 'WÖRTERBUCH' der Maschine 'WORTENTSCHEIDUNG' als Offerten übergeben. 'WORTENTSCHEIDUNG' fordert nun vom Benutzer eine Aussage darüber ab, ob er die Offerten verwenden möchte. Wenn der Benutzer die offerierten Wortformen in seine Begriffskonstrukte einfügen möchte, wird das Teilsystem 'EDITIEREN' von der Maschine 'WORTENTSCHEIDUNG' generiert und aktiviert. Das Teilsystem 'EDITIEREN' unterstützt den Benutzer beim Korrigieren seiner Begriffskonstrukte.

Der Informationsverarbeitungsprozess zur Ermittlung von Offerten endet mit einem der folgenden Ereignisse:

Ende 1: Die Maschine 'DOKUMENTENANALYSE' erhält von der Maschine 'IRS' die Information, dass aus dem Selektionsergebnis kein weiteres Dokument mehr bereitgestellt werden kann.

Ende 2: Die Maschine 'WORTENTSCHEIDUNG' erhält vom Benutzer die Aufforderung, ihm keine weiteren Wortformen mehr anzubieten.

In jedem Fall erhält der 'INITIATOR', der ja der Generator beider Maschinen 'DOKUMENTENANALYSE' und 'WORTENTSCHEIDUNG' ist, die Steuerung zurück. Der 'INITIATOR' veranlasst daraufhin das Teilsystem 'WÖRTERBUCH', das Filtern zu beenden, und löscht dann alle verwendeten

Maschinen des Teilsystems 'BEGRIFFSKONSTRUKTE PRÄZISIEREN'. Dann gibt er die Steuerung an die Maschine 'SUCHANFRAGE' zurück.

Das Teilsystem 'WÖRTERBUCH' ist kein Bestandteil des Maschinensystems 'BEGRIFFSKONSTRUKTE PRÄZISIEREN'. Das hat seinen Grund darin, dass das 'WÖRTERBUCH' als ein lernendes System mit einem eigenen „Gedächtnis“ konzipiert ist. Es existierte bereits vor der Aktivierung des Teilsystems 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' und soll es auch „überleben“, um die erworbene Fähigkeit zur Wortnormierung für den nachfolgenden überwachten Lernprozess zu bewahren.

5.5 Das Teilsystem 'SUCHANFRAGE KONSTRUIEREN'

5.5.1 Problemstellung

Die Aufgabe, die das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' zu lösen hat, ergibt sich aus folgender Situation: Der Benutzer hat zunächst seine Begriffskonstrukte deklariert und möchte sich nun bei der Konstruktion der Suchanfrage zu seinem Thema vom KI-Assistenten unterstützen lassen.

Die Begriffskonstrukte bringen schon viele Aspekte des Themas zum Ausdruck. Unklar ist jedoch noch, in welcher Weise sie zu verknüpfen sind. Dies soll in einem überwachten Lernprozess geklärt werden, in dem die Begriffskonstrukte als „Vorerfahrung“ verwendet werden. Das Ziel dieses Lernprozesses ist es, in Zusammenarbeit mit dem Benutzer einen Klassifikator aufzubauen, der mit akzeptabler Vollständigkeit und ausreichender Genauigkeit relevante Dokumente selektiert.

Der gesuchte Klassifikator wird in zwei Stufen aufgebaut: Zunächst wird eine ***Grobanfrage*** formuliert, die eine hohe Vollständigkeit des Selektionsergebnisses sichert. Dann wird die Menge der Dokumente, die der Grobanfrage genügen, durch eine ***Feinanfrage*** soweit eingeschränkt, dass auch die Genauigkeit akzeptabel wird. Diese Zweistufigkeit entspricht der Strategie, eine notwendige Bedingung durch eine hinreichende Bedingung zu präzisieren.

Wir beginnen mit der Konstruktion der Grobanfrage und gehen dabei von der Annahme aus, dass Dokumente, die nicht wenigstens einem der deklarierten Begriffskonstrukte genügen, für das Thema nicht relevant sein können. Die

Grobanfrage könnte somit in einfacher Weise durch die disjunktive Verknüpfung aller Begriffskonstrukte gebildet werden. Nun ist aber zu berücksichtigen, dass der Benutzer mitunter Begriffskonstrukte ausschließlich zu dem Zweck deklariert, um sie in anderen Begriffskonstrukten referieren zu können. Derartige „Hilfskonstrukte“ haben dann hinsichtlich des Themas keine eigenständige Bedeutung.

Die vom Benutzer deklarierten Begriffskonstrukte bilden die Menge R:

$$R = \left\{ r_1, r_2, ..., r_k \right\} \tag{50}$$

Ein Begriffskonstrukt kann gemäß Grammatik G_B Referenzen auf andere Begriffskonstrukte enthalten. Wir beschreiben diesen Sachverhalt durch eine Relation ρ_R und erhalten damit das System Γ der Begriffskonstrukte:

$$\Gamma = (R, \rho_R) \quad \text{mit} \quad \rho_R \subset R \times R \tag{51}$$

Ein Element $(r_i, r_j) \in \rho_R$ bringt dabei zum Ausdruck, dass das Begriffskonstrukt r_i das Begriffskonstrukt r_j referiert. Da gemäß einer Kontextbedingung zur Grammatik G_B für jedes Element $(r_i, r_j) \in \rho_R$ stets $i > j$ gelten muss, bildet das System Γ einen zyklenfreien Graphen. Die Menge R' seiner „Eintrittspunkte“ ist:

$$R' = \left\{ r_j \mid r_j \in R \ \wedge \ \neg \exists (r_i \in R) \left[(r_i, r_j) \in \rho_R \right] \right\} \tag{52}$$

Sie enthält also alle diejenigen Begriffskonstrukte, die von keinem anderen Begriffskonstrukt referiert werden.

Jedes Begriffskonstrukt $r \in R'$ wird in eine Suchanfrage S_r transformiert, die der Kommandosprache des Information-Retrieval-Systems STAIRS entspricht. Durch die disjunktive Verknüpfung aller dieser Suchanfragen wird die Grobanfrage S_0 gewonnen:

$$S_0 = \bigvee_{r \in R'} S_r \tag{53}$$

Die Grobanfrage S_0 wird an das Information-Retrieval-System STAIRS gerichtet. STAIRS arbeitet sie in der Datenbank ab und stellt ein Selektionsergebnis bereit, das den Ausgangspunkt für die Konstruktion der Feinanfrage bildet.

Durch das Information-Retrieval-System STAIRS wird ein Selektionsergebnis als eine Menge von ***Fundorten*** dargestellt. Ein Fundort φ wird dabei durch vier Koordinaten beschrieben:

$$\varphi = (\kappa_1, \kappa_2, \kappa_3, \kappa_4) \tag{54}$$

Die vier Koordinaten haben folgende Bedeutung:

κ_1: Laufende Nummer des Dokuments in der Datenbank,

κ_2: Identifikator des Segments innerhalb des Dokuments,

κ_3: Laufende Nummer des Satzes innerhalb des Segments,

κ_4: Laufende Nummer des Wortes innerhalb des Satzes.

Die detaillierteste Beschreibung eines Fundortes liegt dann vor, wenn er in allen vier Koordinaten bestimmt ist. Auf der Ebene der Begriffskonstrukte ist das im Falle von STAIRS stets gegeben. Durch Selektionsoperationen kann eine schrittweise „Vergröberung" der Beschreibung eines Fundortes eintreten. Wenn dann schließlich Dokumente ausgegeben werden sollen, ist nur noch die erste Koordinate des Fundortes von Interesse.

Da jedoch auch bei einer detaillierteren Beschreibung der Fundorte häufig Aussagen über Dokumente zu treffen sind, führen wir dafür eine spezielle Sprechweise ein: Liegt in einer Menge von Fundorten ein solcher Fundort, dessen erste Koordinate auf ein bestimmtes Dokument verweist, so werden wir sagen, dass dieses Dokument in der Fundortmenge ***referiert*** wird.

Wir werden im weiteren durch die Angabe eines Niveaus ν ausdrücken, in wie viel Koordinaten ein Fundort bestimmt ist:

$$\varphi^{\nu} = \left(\kappa_1^{\nu}, \kappa_2^{\nu}, \kappa_3^{\nu}, \kappa_4^{\nu}\right); \quad \kappa_i^{\nu} = \otimes \text{ für } i > \nu \tag{55}$$

Ein Fundort auf dem Niveau ν ist nur in seinen ersten ν Koordinaten bestimmt - die restlichen Koordinaten sind unbestimmt. Diese Unbestimmtheit drücken wir durch das Symbol "$\otimes$" aus. Wird im Zuge von Selektionsoperationen eine „Vergröberung" der Fundortbeschreibung von bisher ν auf nunmehr $\nu' < \nu$ Koordinaten vorgenommen, so entspricht das einer Projektion des Fundortes aus dem ν-dimensionalen Raum auf den ν'-dimensionalen Raum. Wir beschreiben diese Projektion mit Hilfe einer Projektionsfunktion $\boldsymbol{\mathcal{P}}^{\nu'}$:

$$\begin{aligned} \boldsymbol{\mathcal{P}}^{\nu'}\left(\varphi^{\nu}\right) &= \left(\kappa_1^{\nu'}, \kappa_2^{\nu'}, \kappa_3^{\nu'}, \kappa_4^{\nu'}\right) \\ \kappa_i^{\nu'} &= \begin{cases} \kappa_i^{\nu} & i \le \nu' \\ \otimes & i > \nu' \end{cases} \end{aligned} \tag{56}$$

Die Menge aller Fundorte, die in einem Selektionsprozess ermittelt wurden, symbolisieren wir durch $\mathcal{F}_S^\nu$. Die Fundorte in dieser Menge stimmen in zwei Merkmalen überein:

1. im Niveau ν, auf dem der Fundort beschrieben ist, und

2. in der Folge S von Selektionsoperationen, in deren Ergebnis der Fundort ermittelt wurde.

Die Menge der Fundorte, die durch die Grobanfrage S_0 selektiert wurden, wird durch $\mathcal{F}_{S_0}^4$ symbolisiert. Sie enthält Fundortbeschreibungen, die auf dem Wortniveau ($\nu = 4$) bestimmt sind.

Mit $\mathcal{F}_{S_0}^4$ liegt ein Selektionsergebnis vor, das mit einer hohen Vollständigkeit Fundorte zum Thema des Benutzers enthält. Die nächste Aufgabe besteht nun darin, die Genauigkeit der Selektion zu erhöhen. Das erfolgt mit Hilfe der ***Feinanfrage***. Die Feinanfrage wird in einem arbeitsteiligen Informationsverarbeitungsprozess konstruiert, in dem der Benutzer und der KI-Assistent spezifische Rollen übernehmen:

- Der Benutzer beschreibt den Inhalt seines Themas, indem er Dokumente hinsichtlich ihrer Relevanz bewertet. In Analogie zum Begriffslernen (Lernen von Konzepten) gibt er damit eine ***extensionale Bestimmung*** des Themas.[1]

- Durch Auswertung der extensionalen Bestimmung des Themas unterstützt der KI-Assistent den Benutzer bei der Formulierung der ***Intension*** seines Themas. Der KI-Assistent nutzt dazu die in den Begriffskonstrukten enthaltene „Vorerfahrung". Einerseits ermittelt er Wortformen, die geeignet erscheinen, die Begriffskonstrukte zu präzisieren, andererseits unterbreitet er dem Benutzer Vorschläge, wie er durch Verknüpfung seiner Begriffskonstrukte die Genauigkeit des Selektionsergebnisses erhöhen kann.

[1] Vgl. beispielsweise Beierle/Kern-Isberner (2000), S. 116 ff.

Das Problem, aus der extensionalen Bestimmung eines Themas dessen Intension automatisch abzuleiten, ist mit Algorithmen des induktives Lernens (Begriffslernens) lösbar.[1] Experimente mit lernfähigen Klassifizierungssystemen haben gezeigt, dass diese Algorithmen auch für die automatisierte Konstruktion von Suchanfragen an ein Information-Retrieval-System geeignet sind.[2]

In einer Reihe von Methoden, die zur Konstruktion von Klassifikatoren relevanzbewertete Dokumente analysieren, werden lineare Schwellenwert-Elemente verwendet.[3] Die Autoren dieser Untersuchungen kritisieren, dass die gängigen Information-Retrieval-Systeme jedoch mit den Möglichkeiten ihrer Anfragesprache nur logische Anfrageformulierungen verarbeiten können.[4] Um die Passfähigkeit zu STAIRS zu sichern, beschränken wir uns auf die Konstruktion graphentheoretisch realisierter Klassifikatoren, die sich auf logische Ausdrücke abbilden lassen.

Im Lernprozess zur Konstruktion eines Klassifikators sind drei Phasen zu unterscheiden:

1. ***Belehrungsphase:*** Aus der Gesamtheit der zu klassifizierenden Objekte wird eine Teilmenge entnommen, die als ***Lernmenge*** bezeichnet wird. Für jedes Objekt dieser Lernmenge gibt der Benutzer explizit die Klassenzugehörigkeit an.

2. ***Lernphase:*** Durch eine Analyse der Objekte mit bekannter Klassenzugehörigkeit wird ein ***Klassifikator*** konstruiert.

3. ***Kannphase:*** Der ermittelte Klassifikator wird auf Objekte angewendet, deren Klassenzugehörigkeit unbekannt ist.

Bei den meisten Lernverfahren wird die Lernmenge vor Beginn der Lernphase gebildet.[5] Diese Vorgehensweise ist deshalb problematisch, weil gewöhnlich vor Beginn der Lernphase noch nicht bekannt ist, welche Objekte in die Lernmenge aufzunehmen sind. Wir lösen dieses Problem, indem wir Belehrungsphase, Lernphase und Kannphase verschränkt ablaufen lassen.

---

[1] Vgl. beispielsweise Russell/Norvig (2004), S. 796 ff.
[2] Vgl. Jarosch/Müller (1984).
[3] Vgl. beispielhaft Blosseville/Hebrail/Monteil/Penot (1992), Chen (1995) und Bishop (2006).
[4] Vgl. beispielsweise Sormunen (2001).
[5] Vgl. beispielsweise Unger/Wysotzki (1981).

5.5.2 Methode für die Konstruktion des Klassifikators

Durch Auswertung der Grobanfrage S_0 wurde in einer ersten Stufe die Fundortmenge $\mathcal{F}_{S_0}^4$ gewonnen, in der mit hoher ***Vollständigkeit*** relevante Dokumente referiert werden. Diese Fundortmenge ist nun in einer zweiten Stufe derart einzuschränken, dass die ***Genauigkeit*** des Recherche-Ergebnisses erhöht wird.

Dieses Ziel wird erreicht, indem durch eine Folge von Selektionsoperationen die Menge $\mathcal{F}_{S_0}^4$ sukzessive in Teilmengen zerlegt wird. Die Zerlegungs-Schritte werden dabei so gestaltet, dass letztlich einerseits „relevante“ Teilmengen mit Fundorten aus hauptsächlich relevanten Dokumenten und andererseits „irrelevante“ Teilmengen mit Fundorten aus hauptsächlich irrelevanten Dokumenten gewonnen werden. Dann können jene Dokumente von der Ausgabe ausgeschlossen werden, die in den „irrelevanten“ Teilmengen referiert werden.

Jeder Zerlegungs-Schritt geht von einer Fundortmenge aus, für die entschieden werden muss, in welcher Weise sie weiter zu behandeln ist. Diese Menge nennen wir ***Objektmenge***.

Für eine Objektmenge kann eine der folgenden Entscheidungen gefällt werden:

1. Alle in der Objektmenge referierten Dokumente werden der Klasse „***ausgeben***“ zugeordnet. Sie gehören dann zum Selektionsergebnis der präzisierten Suchanfrage.

2. Alle in der Objektmenge referierten Dokumente werden der Klasse „***nicht ausgeben***“ zugeordnet. Sie werden dann in den weiteren Entscheidungsprozessen nicht mehr berücksichtigt. Sie gehören nicht zum Selektionsergebnis der präzisierten Suchanfrage.

3. Durch eine ***Selektionsoperation*** wird die Objektmenge zunächst in zwei Teilmengen zerlegt, die dann einzeln weiterbehandelt werden. Bei der Auswahl einer zweckmäßigen Selektionsoperation wird der Benutzer durch den KI-Assistenten unterstützt. Dazu muss der Benutzer Dokumente, die in der Objektmenge referiert werden, hinsichtlich ihrer Relevanz bewerten. Die dadurch entstehende Menge von bewerteten Fundorten bezeichnen wir als ***Lernmenge***.

Trifft der Benutzer die Entscheidung, dass eine Selektionsoperation ausgeführt werden soll, so bildet der KI-Assistent vier Mengen von Fundorten:

- die ***selektierte Objektmenge:*** Sie enthält alle Fundorte der Objektmenge, die durch die Selektionsoperation ermittelt wurden;
- die ***Rest-Objektmenge:*** Sie enthält alle Fundorte der Objektmenge, die nicht durch die Selektionsoperation ermittelt wurden;
- die ***selektierte Lernmenge:*** Sie enthält alle Fundorte der Lernmenge, die durch die Selektionsoperation ermittelt wurden;
- die ***Rest-Lernmenge:*** Sie enthält alle Fundorte der Lernmenge, die nicht durch die Selektionsoperation ermittelt wurden.

Die durch die Selektionsoperation bewirkten Mengen-Zerlegungen sind in Abbildung 18 dargestellt.

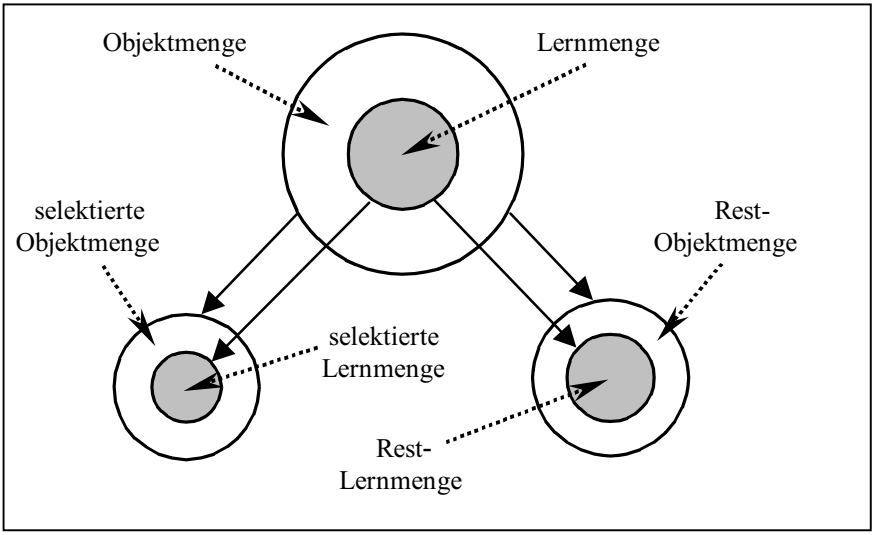


Abbildung 18: Zerlegung der Objektmenge und der Lernmenge durch eine Selektionsoperation

Nach der Formulierung und Abarbeitung der Grobanfrage S_0 setzt das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' die Konstruktion des Klassifikators mit der Behandlung der Objektmenge $\mathcal{F}^4_{S_0}$ fort. Durch die Relevanzbewertung von Dokumenten, die in $\mathcal{F}^4_{S_0}$ referiert werden, bildet der Benutzer die Lernmenge $f^4_{S_0}$. Entscheidet sich der Benutzer dann für eine Selektionsoperation σ, so werden die folgenden Teilmengen gebildet:

- die selektierte Objektmenge: $\mathcal{F}^{\nu'}_{S_0\sigma}$,
- die Rest-Objektmenge: $\mathcal{F}^4_{S_0\bar{\sigma}}$,
- die selektierte Lernmenge: $f^{\nu'}_{S_0\sigma}$ und
- die Rest-Lernmenge: $f^4_{S_0\bar{\sigma}}$.

Mit der Deklaration seiner Begriffskonstrukte hat der Benutzer die Voraussetzungen dafür geschaffen, dass nach Aussagen gesucht werden kann. Die Aussagen können auf dem Satz-, Segment- oder Dokumentenniveau gesucht werden. Entsprechend liegt das Selektionsergebnis auf einem dieser drei Niveaus: Der Index ν' nimmt die Werte 3, 2 oder 1 an.

Die Rest-Objektmenge und die Rest-Lernmenge enthalten jene Fundorte der Ausgangsmengen, die ***nicht*** durch σ selektiert wurden. Sie verbleiben folglich auf dem Ausgangniveau $\nu = 4$.

In den nächsten Zerlegungs-Schritten müssen die entstandenen Teilmengen in gleicher Weise behandelt werden wie die Ausgangsmenge - das Verfahren zur Konstruktion der zweiten Stufe des Klassifikators ist also ***rekursiv***.

Bei der Beschreibung der Methode für die Konstruktion des Klassifikators werden wir in folgenden Schritten vorgehen:

1. Im Abschnitt 5.5.2.1 beschreiben wir zunächst, in welcher Weise ein ***allgemeiner Rekursions-Schritt*** des Verfahrens abläuft.

2. Dann stellen wir im Abschnitt 5.5.2.2 den ***Gesamtprozess*** der Konstruktion des Klassifikators dar.

3. Der Abschnitt 5.5.2.3 befasst sich mit der ***Optimierung*** der zweiten Stufe des Klassifikators.

4. Schließlich wird im Abschnitt 5.5.3 im Rahmen der Beschreibung des Maschinensystems 'SUCHANFRAGE KONSTRUIEREN' erläutert, wie der optimierte Klassifikator auf die ***Anfragesprache*** des Information-Retrieval-Systems STAIRS abgebildet wird.

5.5.2.1 Ein allgemeiner Rekursions-Schritt

Es sei $\mathcal{F}_{\tilde{p}}^{\nu}$ die zu behandelnde Objektmenge von Fundorten. Der Index $\tilde{p}$ gibt die Folge von Selektionsoperationen an, durch die die Objektmenge aus der Datenbank selektiert wurde. Er hat die Form:

$$\begin{aligned} &\tilde{p} = S_0 \hat{\sigma}_1 \hat{\sigma}_2 \dots \hat{\sigma}_k \\ &\hat{\sigma}_i \in \{ \sigma_i, \overline{\sigma}_i \}; \qquad i = 1, 2, \dots, k \end{aligned} \tag{57}$$

Wir nehmen an, der Benutzer habe bereits einige der in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente hinsichtlich ihrer Relevanz bewertet. Die Fundortangaben aus diesen relevanzbewerteten Dokumenten bilden die Lernmenge $\mathcal{f}_{\tilde{p}}^{\nu}$. Sie ist eine Teilmenge von $\mathcal{F}_{\tilde{p}}^{\nu}$:

$$\mathcal{F}_{\tilde{p}}^{\nu} = \left\{ \begin{array}{l} \varphi^{\nu} = \left(\kappa_1^{\nu}, \kappa_2^{\nu}, \kappa_3^{\nu}, \kappa_4^{\nu}\right) \mid \\ \qquad \varphi^{\nu} \in \mathcal{F}_{\tilde{p}}^{\nu} \wedge \text{Dokument } \kappa_1^{\nu} \text{ wurde bewertet} \end{array} \right\} \tag{58}$$

Die Relevanzentscheidung, die der Benutzer für ein bewertetes Dokument getroffen hat, wird durch die Funktion Rel angegeben:

$$\text{Rel}\left(\kappa_1^{\nu}\right) = \left\{ \begin{array}{ll} +1 & \text{Dokument } \kappa_1^{\nu} \text{ ist relevant} \\ -1 & \text{Dokument } \kappa_1^{\nu} \text{ ist irrelevant} \end{array} \right. \tag{59}$$

Der auszuführende Rekursions-Schritt hat die Aufgabe, eine Entscheidung über die Behandlung der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ herbeizuführen. Diese Entscheidung trifft der Benutzer durch zwei Formen von Anweisungen:

1. ***Globale Anweisung:*** Der Benutzer gibt eine Regel an, nach der die Entscheidung automatisch gefällt wird. Dabei formuliert er zugleich die Bedingungen, unter denen diese Regel anwendbar ist.

2. ***Individuelle Anweisung:*** In allen Situationen, in denen nicht nach der globalen Regel entschieden werden kann, wird dem Benutzer eine individuelle Entscheidung abverlangt.

5.5.2.1.1 Zuordnung der gesamten Objektmenge zu einer Klasse

Bei der Behandlung der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ wird zunächst geprüft, ob auf eine weitere Zerlegung der Objektmenge verzichtet werden kann. Das ist dann der Fall, wenn in $\mathcal{F}_{\tilde{p}}^{\nu}$ entweder fast nur relevante oder kaum relevante Dokumente referiert werden. Dann können nämlich alle referierten Dokumente

geschlossen entweder der Klasse „ausgeben“ oder der Klasse „nicht ausgeben“ zugeordnet werden.

Wird diese Entscheidung automatisch gefällt, so muss der Benutzer mit zwei Fehlerarten rechnen:

1. Gemeinsam mit den relevanten Dokumenten werden auch Dokumente ausgegeben, die irrelevant sind. Der Benutzer muss in diesem Fall ***Ballast*** akzeptieren.

2. Gemeinsam mit den irrelevanten Dokumenten werden auch relevante Dokumente zurückgehalten. Der Benutzer muss in diesem Fall ***Verlust*** in Kauf nehmen.

Der Benutzer muss die Fehlergrenzen vorgeben, die er noch akzeptiert. Diese Grenzwerte beziehen sich auf den Anteil der in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten relevanten Dokumente an der Gesamtzahl aller in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente. Dieser Anteil kann nur dann exakt bestimmt werden, wenn für alle in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente eine Relevanzbewertung durchgeführt wurde. Im allgemeinen ist das jedoch nicht der Fall. Dann muss aus der Kenntnis der Verhältnisse in der Lernmenge $f_{\tilde{p}}^{\nu}$ auf die Verhältnisse in der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ geschlossen werden. Mit Hilfe statistischer Verfahren wird die Wahrscheinlichkeit geschätzt, dass ein Dokument, das in $\mathcal{F}_{\tilde{p}}^{\nu}$ referiert wird, relevant ist. Diese Wahrscheinlichkeit liegt bei Berücksichtigung einer vom Benutzer vorgegebenen Irrtumswahrscheinlichkeit α in einem Intervall, dessen Grenzen p_u^r und p_o^r im folgenden berechnet werden.

Wir bestimmen zunächst die Anzahl $n_{\tilde{p}}$ ***aller*** in der Lernmenge $f_{\tilde{p}}^{\nu}$ referierten Dokumente

$$n_{\tilde{p}} = \left| \left\{ \mathcal{P}^1\left(\varphi^\nu\right) \middle| \varphi^\nu \in f_{\tilde{p}}^\nu \right\} \right| \tag{60}$$

sowie die Anzahl $n_{\tilde{p}}^r$ der in der Lernmenge $f_{\tilde{p}}^\nu$ referierten ***relevanten*** Dokumente:

$$n_{\tilde{p}}^r = \left| \left\{ \mathcal{P}^1\left(\varphi^\nu\right) \middle| \begin{array}{l} \varphi^\nu = \left(\kappa_1^\nu, \kappa_2^\nu, \kappa_3^\nu, \kappa_4^\nu\right) \in f_{\tilde{p}}^\nu \ \wedge \\ \mathrm{Rel}(\kappa_1^\nu) = +1 \end{array} \right\} \right| \tag{61}$$

Die Anzahl $N_{\tilde{p}}$ aller in der Objektmenge $\mathcal{F}_{\tilde{p}}^\nu$ referierten Dokumente ergibt sich aus

$$N_{\tilde{p}} = \left| \left\{ \mathcal{P}^1\left(\varphi^\nu\right) \middle| \varphi^\nu \in \mathcal{F}_{\tilde{p}}^\nu \right\} \right| \tag{62}$$

Aus der relativen Häufigkeit

$$h^r = \frac{n_{\tilde{p}}^r}{n_{\tilde{p}}} \tag{63}$$

des Ereignisses, dass ein in der „Stichprobe“ $f_{\tilde{p}}^\nu$ referiertes Dokument relevant ist, lassen sich unter Beachtung der Endlichkeitskorrektur[1] die Grenzen p_u^r

[1] Vgl. Sachs (1972), S. 262.

und p_o^r dafür berechnet, dass ein in der „Grundgesamtheit“ $\mathcal{F}_{\tilde{p}}^{\nu}$ referiertes Dokument relevant ist:

$$p_{u,o}^r = \left(h^r - \frac{1}{2n_{\tilde{p}}} \right) \pm q \sqrt{\frac{h^r\left(1-h^r\right)}{n_{\tilde{p}}} \; \frac{N_{\tilde{p}} - n_{\tilde{p}}}{N_{\tilde{p}} - 1}} \tag{64}$$

Dabei ist q das Quantil der Normalverteilung, das entsprechend der Irrtumswahrscheinlichkeit α bestimmt wird, die vom Benutzer festzulegen ist.

Die Entscheidung dafür, dass sämtliche in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente der Klasse „ausgeben“ zuzuordnen sind, wird dann automatisch gefällt, wenn gilt:

$$p_{max}^r < p_u^r \tag{65}$$

Die Entscheidung dafür, dass sämtliche in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente der Klasse „nicht ausgeben“ zuzuordnen sind, wird automatisch gefällt, wenn gilt:

$$p_o^r < p_{min}^r \tag{66}$$

Die beiden Schranken p_{min}^r und p_{max}^r werden vom Benutzer vorgegeben. Sie werden gewöhnlich so gewählt, dass das Intervall $\left[0, p_{min}^r\right]$ kleiner ist als das Intervall $\left[p_{max}^r, 1\right]$. Das liegt daran, dass im allgemeinen für den Benutzer Ballast akzeptabler ist als Verlust. Diese Situation wird durch die Abbildung 19 veranschaulicht.

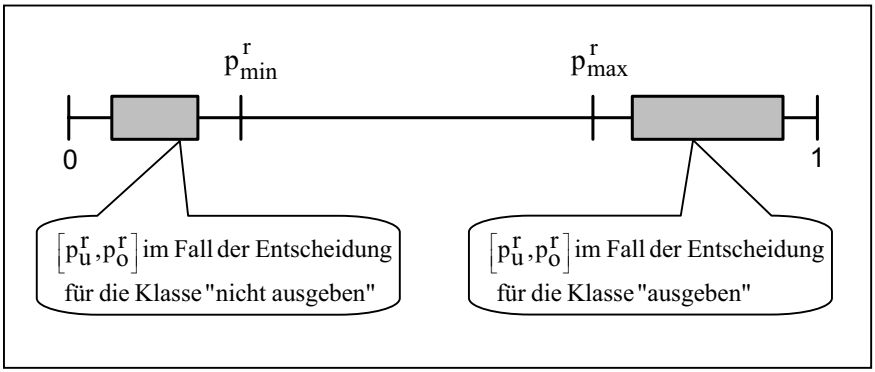


Abbildung 19: Wahrscheinlichkeitsintervalle im Fall der automatischen Entscheidung für eine der beiden Klassen „ausgeben“ bzw. „nicht ausgeben“

Ist für die Objektmenge $\mathcal{F}^{\nu}_{\tilde{p}}$ weder (65) noch (66) erfüllt, so können die in $\mathcal{F}^{\nu}_{\tilde{p}}$ referierten Dokumente nicht automatisch einer der Klassen „ausgeben“ bzw. „nicht ausgeben“ zugeordnet werden. Eine solche Zuordnung kann dann nur der Benutzer selbst anweisen.

5.5.2.1.2 Ermittlung der zweckmäßigsten Zerlegung

Sind die in der Objektmenge $\mathcal{F}^{\nu}_{\tilde{p}}$ referierten Dokumente weder durch eine Regel noch durch die Entscheidung des Benutzers geschlossen einer der beiden Klassen „ausgeben“ bzw. „nicht ausgeben“ zugeordnet worden, so muss die Objektmenge weiter zerlegt werden. Durch diese Zerlegung sollen die relevanten Dokumente möglichst gut von den irrelevanten getrennt werden. Es muss also diejenige Zerlegung gesucht werden, die in diesem Sinne die zweckmäßigste ist. Dazu müssen sämtliche Zerlegungen der Objektmenge $\mathcal{F}^{\nu}_{\tilde{p}}$, die mit den Begriffskonstrukten der Suchanfrage realisierbar sind, auf ihre Zweckmäßigkeit hin untersucht werden.

Eine Zerlegung erfolgt, indem mit Hilfe einer Selektionsoperation σ eine Teilmenge von Fundorten aus $\boldsymbol{\mathcal{F}}_{\tilde{p}}^{\nu}$ selektiert wird, wobei eine Restmenge von Fundorten übrigbleibt. Eine Aussage über die Zweckmäßigkeit der Zerlegung, die durch eine gegebene Selektionsoperation σ erreicht wird, kann lediglich in der Lernmenge $\boldsymbol{f}_{\tilde{p}}^{\nu}$ getroffen werden, denn nur dort liegen relevanzbewertete Dokumente vor.

Eine Selektionsoperation σ ist ein Tripel $\sigma = (\xi, r, \nu')$, in dem das Signum ξ, das Begriffskonstrukt r und das Ergebnisniveau ν' zusammengefasst sind.

Bei der Ausführung der Selektionsoperation σ spielen alle jene Fundorte aus der Lernmenge eine Rolle, die dem Begriffskonstrukt $r \in R'$ genügen. Die Menge dieser Fundorte kann durch die Anwendung der Suchanfrage S_r auf die Lernmenge $\boldsymbol{f}_{\tilde{p}}^{\nu}$ gebildet werden:

$$\boldsymbol{f}_{\tilde{p}S_r}^{\nu} = \left\{ \varphi^{\nu} \,\middle|\, \varphi^{\nu} \in \boldsymbol{f}_{\tilde{p}}^{\nu} \wedge \text{am Fundort } \varphi^{\nu} \text{ ist } S_r \text{ erfüllt} \right\} \qquad (67)$$

Das Signum $\xi \in \{+1, -1\}$ gibt an, ob das Begriffskonstrukt r dazu verwendet werden soll, um „den Begriff zu suchen“ oder um „den Begriff auszuschließen“. Die genaue Semantik des Signums ξ wird später aus der formalen Beschreibung der Selektionsoperation ersichtlich werden.

Das Ergebnisniveau $\nu' \in \{1, 2, 3\}$ legt fest, in wie viel Koordinaten die Fundorte des Selektionsergebnisses bestimmt sind. Es gibt also an, ob die Selektion auf dem Satz-, dem Segment- oder dem Dokumentenniveau stattfinden soll. Da die Fundorte der Lernmenge $\boldsymbol{f}_{\tilde{p}}^{\nu}$ in ν Koordinaten bestimmt sind, muss gelten: $\nu' \leq \nu$.

Durch die Anwendung der Selektionsoperation $\sigma = (\xi, r, \nu')$ auf die Lernmenge $f_{\tilde{p}}^{\nu}$ erhalten wir die selektierte Lernmenge $f_{\tilde{p}\sigma}^{\nu'}$:

$$f_{\tilde{p}\sigma}^{\nu'} = (\xi, r, \nu') \circ f_{\tilde{p}}^{\nu} = \left\{ \mathcal{P}^{\nu'}\left(\varphi^{\nu}\right) \mid \varphi^{\nu} \in f_{\tilde{p}}^{\nu} \wedge B_{\xi r \nu'}\left(\varphi^{\nu}\right) \right\} \quad (68)$$

Die selektierte Lernmenge $f_{\tilde{p}\sigma}^{\nu'}$ enthält die Projektionen auf ν' Koordinaten jener Fundorte aus $f_{\tilde{p}}^{\nu}$, die die Selektionsbedingung $B_{\xi r \nu'}$ erfüllen:

$$B_{\xi r \nu'}\left(\varphi^{\nu}\right) = \begin{cases} \exists \left(\psi^{\nu} \in f_{\tilde{p} S_r}^{\nu}\right)\left[\mathcal{P}^{\nu'}\left(\varphi^{\nu}\right) = \mathcal{P}^{\nu'}\left(\psi^{\nu}\right)\right] & \xi = +1 \\ \forall \left(\psi^{\nu} \in f_{\tilde{p} S_r}^{\nu}\right)\left[\mathcal{P}^{\nu'}\left(\varphi^{\nu}\right) \neq \mathcal{P}^{\nu'}\left(\psi^{\nu}\right)\right] & \xi = -1 \end{cases} \quad (69)$$

Im Falle eines positiven Signums ($\xi = +1$) wird die Menge $f_{\tilde{p} S_r}^{\nu}$ zum Begriffskonstrukt r verwendet, um aus $f_{\tilde{p}}^{\nu}$ jene Fundorte zu selektieren, die auch in $f_{\tilde{p} S_r}^{\nu}$ enthalten sind. Im Falle eines negativen Signums ($\xi = -1$) wird sie verwendet, um aus $f_{\tilde{p}}^{\nu}$ jene Fundorte zu selektieren, die nicht zugleich auch in $f_{\tilde{p} S_r}^{\nu}$ enthalten sind. In beiden Fällen werden nur die ersten ν' Koordinaten der Fundorte in den Vergleich einbezogen.

Unter Verwendung der selektierten Lernmenge $f_{\tilde{p}\sigma}^{\nu'}$ können wir nun die Rest-Lernmenge bilden:

$$\boldsymbol{f}_{\tilde{p}\bar{\sigma}}^{\nu} = \left\{ \begin{array}{l} \varphi^{\nu} \mid \varphi^{\nu} \in \boldsymbol{f}_{\tilde{p}}^{\nu} \quad \wedge \\ \qquad \forall \left(\psi^{\nu'} \in \boldsymbol{f}_{\tilde{p}\sigma}^{\nu'} \right) \left[\boldsymbol{\mathcal{P}}^{1} \left(\varphi^{\nu} \right) \neq \boldsymbol{\mathcal{P}}^{1} \left(\psi^{\nu'} \right) \right] \end{array} \right\} \tag{70}$$

Die Rest-Lernmenge $\boldsymbol{f}_{\tilde{p}\bar{\sigma}}^{\nu}$ enthält also nur noch jene Fundorte aus $\boldsymbol{f}_{\tilde{p}}^{\nu}$, die ein Dokument referieren, das nicht in der selektierten Lernmenge $\boldsymbol{f}_{\tilde{p}\sigma}^{\nu'}$ referiert wird. Die Fundorte der Rest-Lernmenge befinden sich stets auf demselben Niveau ν wie die Fundorte der Lernmenge $\boldsymbol{f}_{\tilde{p}}^{\nu}$.

Mit der Bildung von $\boldsymbol{f}_{\tilde{p}\sigma}^{\nu'}$ und von $\boldsymbol{f}_{\tilde{p}\bar{\sigma}}^{\nu}$ haben wir das Ziel erreicht, die Lernmenge $\boldsymbol{f}_{\tilde{p}}^{\nu}$ in die selektierte Lernmenge und in die Rest-Lernmenge zu zerlegen. Die beiden Teilmengen müssen nun einzeln weiterbehandelt werden.

In der von uns beschriebenen Methode wird zuerst die selektierte Lernmenge behandelt, also jene Teilmenge, die durch die Selektionsoperation σ näher intensional bestimmt wurde. Ihre Intension ist durch $\tilde{p}\sigma$ beschrieben. Im Zuge der Weiterbehandlung von $\boldsymbol{f}_{\tilde{p}\sigma}^{\nu'}$ wird - unter Umständen erst in tieferen Rekursions-Schritten - über jedes in $\boldsymbol{f}_{\tilde{p}\sigma}^{\nu'}$ referierte Dokument entschieden, ob es der Klasse „ausgeben“ oder der Klasse „nicht ausgeben“ zuzuordnen ist.

Wenn dann die Behandlung der Rest-Lernmenge $\boldsymbol{f}_{\tilde{p}\bar{\sigma}}^{\nu}$ aufgenommen wird, muss nur noch über jene Dokumente entschieden werden, für die bisher noch keine Entscheidung gefallen ist. Das wird durch die spezielle Bildung der Rest-Lernmenge gemäß (70) gesichert.

Nachdem beschrieben wurde, wie mit Hilfe einer Selektionsoperation σ die selektierte Lernmenge und die Rest-Lernmenge gebildet werden, wenden wir uns nun der Frage zu, welche Zerlegung die zweckmäßigste ist.

Zunächst ist zu klären, welche Selektionsoperationen für die Zerlegung der Lernmenge $f_{\tilde{p}}^{\nu}$ in Frage kommen. Von den Begriffskonstrukten der Menge R' scheiden natürlich jene aus, die schon bei der Bildung der Lernmenge ausgewertet wurden: Die Lernmenge $f_{\tilde{p}}^{\nu}$ wurde durch Ausführung der Folge von Selektionsoperationen $\tilde{p} = S_0 \hat{\sigma}_1 \hat{\sigma}_2 \ldots \hat{\sigma}_k$ gebildet. Dabei gilt: $\hat{\sigma}_i \in \{\sigma_i, \overline{\sigma}_i\}$ mit $\sigma_i = (\xi_i, r_i, \nu_i)$ für $i = 1, 2, \ldots, k$. Die Rest-Menge $R_{\tilde{p}}$ der noch auswertbaren Begriffskonstrukte ist somit:

$$R_{\tilde{p}} = R' - \{r_1, r_2, \ldots, r_k\} \tag{71}$$

Jedes Begriffskonstrukt $r \in R_{\tilde{p}}$ kann mit positivem oder negativem Signum für die Selektion verwendet werden. Wir erhalten somit die Menge

$$\tau_{\tilde{p}} = \{+1, -1\} \times R_{\tilde{p}} \tag{72}$$

Wir betrachten im weiteren ein Element $\boldsymbol{t} \in \tau_{\tilde{p}}$. Mit Hilfe von $\boldsymbol{t}$ lassen sich ν Selektionsoperationen bilden, indem das Niveau des Selektionsergebnisses variiert wird. Wir erhalten die Menge von Selektionsoperationen:

$$\Sigma = \{\Sigma_1, \Sigma_2, \ldots, \Sigma_\nu\}; \; \Sigma_m = (\boldsymbol{t}, m) \text{ für } m = 1, 2, \ldots, \nu \tag{73}$$

Unter Verwendung jeder der Selektionsoperationen $\Sigma_m \in \Sigma$ kann in der Lernmenge $f_{\tilde{p}}^{\nu}$ eine ***Probezerlegung*** in die beiden Teilmengen $f_{\tilde{p}\,\Sigma_m}^{m}$ und $f_{\tilde{p}\,\overline{\Sigma}_m}^{\nu}$ durchgeführt werden. Für jede dieser Probezerlegungen wird geprüft, inwieweit sie der Zielstellung gerecht wird, relevante von irrelevanten Dokumenten zu trennen. Dazu ermitteln wir die Anzahl der Fundorte aus relevanten bzw. irrelevanten Dokumenten in den beiden Teilmengen. Da stets $m \le \nu$ gilt, erfolgt die Zählung der Fundorte in beiden Teilmengen auf dem Niveau m.

Folgende Anzahlen werden bestimmt:

$$
\begin{aligned}
A &= \left| \left\{ \varphi^m \mid \varphi^m = \left(\kappa_1^m, \kappa_2^m, \kappa_3^m, \kappa_4^m\right) \in f_{\tilde{p}\,\Sigma_m}^{m} \wedge \right. \right. \\
&\qquad \left. \left. \mathrm{Rel}\left(\kappa_1^m\right) = +1 \right\} \right| \\
B &= \left| \left\{ \mathcal{P}^m\left(\varphi^\nu\right) \mid \varphi^\nu = \left(\kappa_1^\nu, \kappa_2^\nu, \kappa_3^\nu, \kappa_4^\nu\right) \in f_{\tilde{p}\,\overline{\Sigma}_m}^{\nu} \wedge \right. \right. \\
&\qquad \left. \left. \mathrm{Rel}\left(\kappa_1^\nu\right) = +1 \right\} \right| \\
C &= \left| \left\{ \varphi^m \mid \varphi^m = \left(\kappa_1^m, \kappa_2^m, \kappa_3^m, \kappa_4^m\right) \in f_{\tilde{p}\,\Sigma_m}^{m} \wedge \right. \right. \\
&\qquad \left. \left. \mathrm{Rel}\left(\kappa_1^m\right) = -1 \right\} \right| \\
D &= \left| \left\{ \mathcal{P}^m\left(\varphi^\nu\right) \mid \varphi^\nu = \left(\kappa_1^\nu, \kappa_2^\nu, \kappa_3^\nu, \kappa_4^\nu\right) \in f_{\tilde{p}\,\overline{\Sigma}_m}^{\nu} \wedge \right. \right. \\
&\qquad \left. \left. \mathrm{Rel}\left(\kappa_1^\nu\right) = -1 \right\} \right|
\end{aligned}
\tag{74}
$$

Entsprechend dem Ausdruck (40) wird aus den Anzahlen A, B, C und D der $\hat{\chi}^2$-Wert geschätzt. Er dient als Maß für die Zweckmäßigkeit z_{Σ_m} der Zerlegung mit Hilfe der Selektionsoperation Σ_m:

$$z_{\Sigma_m} = \hat{\chi}^2(A, B, C, D) \tag{75}$$

Für ein gegebenes Element $t \in \tau_{\tilde{p}}$ wird zunächst das Zweckmäßigkeitsmaß z_{Σ_ν} berechnet. Es bezieht sich auf diejenige Selektionsoperation in Σ, die ein Selektionsergebnis auf dem detailliertesten Niveau liefert. Bei der Prüfung, auf welchem Niveau am zweckmäßigsten selektiert werden sollte, lassen wir uns von dem Prinzip leiten, bei der Selektion nur dann einen Niveauverlust in Kauf zu nehmen, wenn dadurch die Zweckmäßigkeit der Zerlegung signifikant ansteigt. Nach diesem Prinzip findet man für das betrachtete Element $t \in \tau_{\tilde{p}}$ das optimale Selektionsniveau ν_t, dem das Zweckmäßigkeitsmaß $z_t \equiv z_{\Sigma_{\nu_t}}$ entspricht.

Das Zweckmäßigkeitsmaß z_t bringt - grob gesprochen - zum Ausdruck, in welchem Maße das Ereignis, dass der Fundort in die selektierte Menge übernommen wird, unabhängig von dem Ereignis ist, dass dieser Fundort in einem relevanten Dokument liegt: Je größer z_t ist, desto unwahrscheinlicher ist die Unabhängigkeit zwischen den beiden Ereignissen.

Die Aufgabe besteht nun darin, unter Verwendung des Zweckmäßigkeitsmaßes diejenige Selektionsoperation σ^* zu ermitteln, die zur „günstigsten“ Zerlegung der Lernmenge $f_{\tilde{p}}^{\nu}$ führt.

Wir prüfen zunächst, ob die Auswahl der Selektionsoperation σ^* mit Hilfe einer globalen Regel automatisch erfolgen kann. Dabei wird folgende globale Regel verwendet:

Regel: Eine Selektionsoperation $(\boldsymbol{t}, \nu_{\boldsymbol{t}})$ kann ohne Bestätigung durch den Benutzer dann automatisch ausgeführt werden, wenn ihr Zweckmäßigkeitsmaß $z_{\boldsymbol{t}}$ einen Schwellenwert z^{a}_{min} überschreitet.

Der Schwellenwert z^{a}_{min} wird vom Benutzer so bemessen. dass er nur von solchen Zerlegungsoperationen überschritten wird, deren Zweckmäßigkeit „außer Zweifel steht".

Die Auswahl der Selektionsoperation σ^{*} kann natürlich nur dann automatisch erfolgen, wenn die Menge

$$\Sigma^{a} = \left\{ (\boldsymbol{t}, \nu_{\boldsymbol{t}}) \mid \boldsymbol{t} \in \tau_{\tilde{p}} \ \wedge \ z_{\boldsymbol{t}} > z^{a}_{min} \right\} \tag{76}$$

wenigstens ein Element enthält. Dann wird σ^{*} als diejenige Selektionsoperation in Σ^{a} ermittelt, für die das Zweckmäßigkeitsmaß sein Maximum annimmt.

Wenn $\Sigma^{a} = \varnothing$ gilt, wenn also die Selektionsoperation σ^{*} nicht automatisch bestimmt werden kann, dann müssen dem Benutzer Selektionsoperationen als Offerten angeboten werden. Zu diesem Zweck bildet das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' die Menge

$$\Sigma^{o} = \left\{ (\boldsymbol{t}, \nu_{\boldsymbol{t}}) \mid \boldsymbol{t} \in \tau_{\tilde{p}} \ \wedge \ z_{\boldsymbol{t}} > z^{o}_{min} \right\} \tag{77}$$

Der Schwellenwert z^{o}_{min} wird vom Benutzer so bemessen, dass ihm hinreichend viele Selektionsoperationen vorgelegt werden. Unter diesen wählt er

diejenige Selektionsoperation σ^* aus, die ihm aus semantischer Sicht am besten zur Präzisierung der intensionalen Beschreibung des Themas geeignet erscheint.

Ist die Menge Σ^o leer oder sind alle angebotenen Selektionsoperationen aus semantischer Sicht unakzeptabel, so hat der Benutzer zwei Möglichkeiten, den Lernprozess fortzusetzen:

1. Durch die Relevanzbewertung weiterer Dokumente aus der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ vergrößert er die Lernmenge $f_{\tilde{p}}^{\nu}$. Er bringt damit gezielt neues „Wissen“ in den überwachten Lernprozess ein und versetzt das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' in die Lage, akzeptablere Selektionsoperationen zu ermitteln. Man sieht, dass die Lernphase verschränkt mit der Belehrungsphase abläuft. Der Benutzer kann damit in Abhängigkeit vom erreichten Stand der intensionalen Beschreibung des Themas neues „Wissen“ in den Lernprozess einbringen.

2. Er gibt die auszuführende Selektionsoperation σ^* selbst an. Dadurch präzisiert er explizit die intensionale Beschreibung des Themas. Er behält sich dabei die Möglichkeit vor, erst in einem spezielleren semantischen Umfeld durch Relevanzbewertung „Wissen“ in den Lernprozess einzubringen.

5.5.2.1.3 Durchführung der Zerlegung

Durch Probezerlegungen der Lernmenge $f_{\tilde{p}}^{\nu}$ wurde die günstigste Selektionsoperation $\sigma^* = (t, \nu_t)$ ermittelt, die zunächst dazu verwendet wird, die Lernmenge $f_{\tilde{p}}^{\nu}$ gemäß (68) - (70) in die selektierte Lernmenge $f_{\tilde{p}\,\sigma^*}^{\nu t}$ und in die Rest-Lernmenge $f_{\tilde{p}\,\bar{\sigma}^*}^{\nu}$ zu zerlegen. Das wird in Abbildung 20 veranschaulicht.

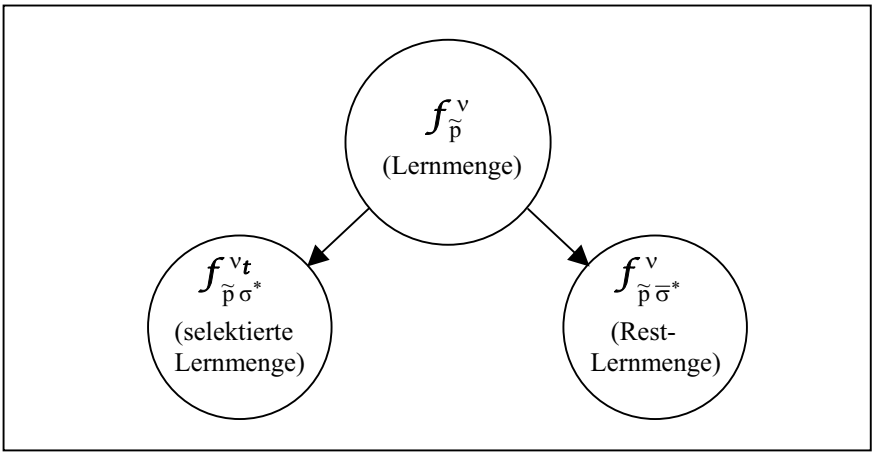


Abbildung 20: Zerlegung der Lernmenge durch die Selektionsoperation σ^*

Danach wird diese Zerlegung auch in der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ nachvollzogen. Die Zerlegung wird damit auch auf solche Fundorte angewendet, die Dokumente referieren, für die ***nicht*** bekannt ist, ob sie der Klasse „ausgeben" oder der Klasse „nicht ausgeben" zuzurechnen sind. Somit läuft die Kannphase parallel zur Lernphase ab.

Während das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' die Zerlegung der Lernmenge $f_{\tilde{p}}^{\nu}$ eigenständig vornimmt, soll die Zerlegung der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ mit den Möglichkeiten des Information-Retrieval-Systems STAIRS erfolgen.

Die Bildung der selektierten Objektmenge $\mathcal{F}_{\tilde{p}\,\sigma^*}^{\nu_t}$ durch Anwendung der Selektionsoperation $\sigma^* = \left(\xi^*, r^*, \nu_t\right)$ gemäß

$$\begin{aligned}\boldsymbol{\mathcal{F}}^{\nu \boldsymbol{t}}_{\tilde{\mathrm{p}}\,\sigma^*} &= \left(\xi^*, \mathrm{r}^*, \nu_{\boldsymbol{t}}\right) \circ \boldsymbol{\mathcal{F}}^{\nu}_{\tilde{\mathrm{p}}} \\ &= \left\{ \boldsymbol{\mathcal{P}}^{\nu \boldsymbol{t}}\left(\varphi^{\nu}\right) \mid \varphi^{\nu} \in \boldsymbol{\mathcal{F}}^{\nu}_{\tilde{\mathrm{p}}} \wedge \mathrm{B}_{\xi^* \mathrm{r}^* \nu_{\boldsymbol{t}}}\left(\varphi^{\nu}\right) \right\}\end{aligned} \tag{78}$$

bereitet dabei keine Probleme, denn diese Operation kann in der Anfragesprache von STAIRS ausgedrückt werden. Wie wir im Abschnitt 5.5.3 zeigen werden, stehen dafür im Falle $\xi^* = +1$ die Operatoren SENT ($\nu_{\boldsymbol{t}} = 3$), SEGM ($\nu_{\boldsymbol{t}} = 2$) bzw. AND ($\nu_{\boldsymbol{t}} = 1$) und im Falle $\xi^* = -1$ die Operatoren NOTSENT ($\nu_{\boldsymbol{t}} = 3$), NOTSEGM ($\nu_{\boldsymbol{t}} = 2$) bzw. NOT ($\nu_{\boldsymbol{t}} = 1$) zur Verfügung.

Problematisch ist dagegen die Bildung der Rest-Objektmenge. Diese müsste in Analogie zur Rest-Lernmenge wie folgt gebildet werden:

$$\boldsymbol{\mathcal{F}}^{\nu}_{\tilde{\mathrm{p}}\,\overline{\sigma}^*} = \left\{ \begin{array}{l} \varphi^{\nu} \mid \varphi^{\nu} \in \boldsymbol{\mathcal{F}}^{\nu}_{\tilde{\mathrm{p}}} \wedge \\ \qquad \forall \left(\psi^{\nu \boldsymbol{t}} \in \boldsymbol{\mathcal{F}}^{\nu \boldsymbol{t}}_{\tilde{\mathrm{p}}\,\sigma^*} \right) \left[\boldsymbol{\mathcal{P}}^{1}\left(\varphi^{\nu}\right) \neq \boldsymbol{\mathcal{P}}^{1}\left(\psi^{\nu \boldsymbol{t}}\right) \right] \end{array} \right\} \tag{79}$$

Sie muss also jene Fundorte auf dem Niveau ν enthalten, die ein Dokument referieren, das nicht zugleich auch in der selektierten Objektmenge referiert wird. Für diese Operation gibt es in STAIRS im Falle $\nu > 1$ keinen Operator. Der geforderte Ausschluss von Fundorten durch alleinige Prüfung der ersten Koordinate (Dokumentennummer) kann in STAIRS nur auf dem Niveau $\nu = 1$ (durch den Operator NOT) erfolgen, wenn die Fundorte also ohnehin nur noch auf dem Dokumentenniveau beschrieben sind.

Bei der Bildung der Rest-Objektmenge haben wir keine andere Wahl als $\boldsymbol{\mathcal{F}}^{\nu}_{\tilde{\mathrm{p}}\,\overline{\sigma}^*} = \boldsymbol{\mathcal{F}}^{\nu}_{\tilde{\mathrm{p}}}$ zu setzen: Wir verwenden also anstelle der eigentlichen Rest-Objektmenge die ursprüngliche unzerlegte Objektmenge. Das ist im Rahmen

von STAIRS die einzige Möglichkeit, einen Verlust an Detailliertheit in der Fundortbeschreibung zu vermeiden.

Nun werden aber in der „verfälschten“ Rest-Objektmenge im allgemeinen auch solche Dokumente referiert, für die der Benutzer bei der Behandlung der selektierten Objektmenge $\mathcal{F}_{\tilde{p}\,\sigma^*}^{\nu t}$ - eventuell in tieferen Rekursions-Schritten - schon eine Klassenentscheidung getroffen hat.

Es kann somit geschehen, dass dem Benutzer aus der „verfälschten“ Rest-Objektmenge bereits bewertete Dokumente erneut zur Entscheidung vorgelegt werden. Das ist inakzeptabel. Deshalb muss immer dann, wenn eine Kommunikation mit dem Benutzer stattfindet, der bei der Zerlegung der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ bewusst begangene Fehler wieder behoben werden.

Dazu werden die Dokumente, deren Klassenzugehörigkeit bereits feststeht und die trotzdem noch in der Rest-Objektmenge referiert werden, zeitweilig ausgeblendet. Daraus ergibt sich die Notwendigkeit, während des gesamten Ablaufs der Rekursions-Schritte zur Konstruktion der zweiten Stufe des Klassifikators in einer Menge $\mathcal{F}^*$ diejenigen Dokumente aufzusammeln, über deren Klassenzugehörigkeit bereits entschieden wurde:

$$\mathcal{F}^* = \left\{ \varphi^1 = \left(\kappa_1^1, \otimes, \otimes, \otimes\right) \mid \begin{array}{l} \text{über } \kappa_1^1 \text{ wurde} \\ \text{bereits entschieden} \end{array} \right\} \tag{80}$$

Ehe dem Benutzer Dokumente zur Relevanzbewertung angezeigt werden, die in der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ referiert werden, bilden wir zunächst die Menge

$$\mathcal{F}_{\tilde{p}}^{1} = \left\{ \begin{array}{l} \mathcal{P}^{1}\left(\varphi^{\nu}\right) \mid \varphi^{\nu} \in \mathcal{F}_{\tilde{p}}^{\nu} \ \wedge \\ \qquad\qquad \forall\left(\psi \in \mathcal{F}^{*}\right)\left[\mathcal{P}^{1}\left(\varphi^{\nu}\right) \neq \psi\right] \end{array} \right\} \tag{81}$$

In $\mathcal{F}_{\tilde{p}}^{1}$ sind alle Referenzen auf jene Dokumente gelöscht, über die bereits entschieden wurde. Der Benutzer erhält zur Relevanzbewertung nur Dokumente der Menge

$$\mathcal{F}_{\tilde{p}}^{*} = \left\{ \begin{array}{l} d \mid \varphi = (d, \otimes, \otimes, \otimes) \in \mathcal{F}_{\tilde{p}}^{1} \ \wedge \\ \qquad \forall\left(\psi \in \mathcal{f}_{\tilde{p}}^{\nu}\right)\left[\mathcal{P}^{1}(\psi) \neq \varphi\right] \end{array} \right\} \tag{82}$$

also Dokumente, die zwar in $\mathcal{F}_{\tilde{p}}^{1}$ referiert werden, für die aber noch keine Relevanzbewertung vorgenommen wurde.

Damit ist der allgemeine Rekursions-Schritt vollständig beschrieben.

5.5.2.1.4 Die durch den allgemeinen Rekursions-Schritt realisierte Intelligenz

Im Rekursions-Schritt zur Behandlung der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ tritt das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' in Kommunikation sowohl mit dem IRS als auch mit dem Benutzer. Das höhere Niveau der Mensch-Maschine-Kommunikation äußert sich darin, dass der KI-Assistent auf der Grundlage eines Lernprozesses Entscheidungen durch Anwendung von Regeln automatisch fällt oder dem Benutzer Entscheidungsalternativen vorschlägt.

Dieser Lernprozess zeichnet sich durch die folgenden qualitativen Merkmale aus:

1. Es wird festgestellt, ob alle in der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente der Klasse „ausgeben“ bzw. der Klasse „nicht ausgeben“ zugeordnet werden können. Diese Entscheidung wird entweder automatisch (durch Regel) oder vom Benutzer gefällt.

2. Können nicht alle in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente derselben Klasse zugeordnet werden, so muss $\mathcal{F}_{\tilde{p}}^{\nu}$ weiter zerlegt werden. Diese Zerlegung erfolgt auf der Grundlage der Selektionsoperation σ^* parallel in der Lernmenge und in der Objektmenge.

3. Die Selektionsoperation σ^* entspricht einer Präzisierung der intensionalen Beschreibung des Themas. Der Benutzer kann diese Präzisierung explizit vornehmen. Er versorgt dadurch das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' mit zusätzlicher „Vorerfahrung“. Während er bisher „Vorerfahrung“ lediglich in Form der Begriffskonstrukte eingebracht hat, dehnt er sie nun auf Angaben zu ihrer Verknüpfung aus.

4. Die Ermittlung der zweckmäßigsten Selektionsoperation erfolgt durch Auswertung der Lernmenge $f_{\tilde{p}}^{\nu}$. Hat der Benutzer durch Relevanzbewertung von Dokumenten aus $\mathcal{F}_{\tilde{p}}^{\nu}$ schon ausreichend viel „Wissen“ über die extensionale Beschreibung seines Themas angereichert, dass nun das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' daraus statistische Hypothesen für die Präzisierung der Intension des Themas ableiten kann, so wird entweder automatisch entschieden oder es werden dem Benutzer Offerten unterbreitet.

5. Da die Zerlegung in der Lernmenge und in der Objektmenge parallel ausgeführt wird, bilden die in der Lernmenge $f_{\tilde{p}}^{\nu}$ referierten Dokumente stets

eine Teilmenge der in der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente. Während in anderen Lernmodellen[1] ein dem Benutzer vorgelegtes Lernobjekt aus einer Gesamtmenge zufällig ausgewählt wird, sind wir in der Lage, ihm jeweils solche Dokumente anzubieten, die dem aktuell erreichten Stand seiner Suchanfrage-Präzisierung entsprechen. Diese werden nämlich der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu};\ \tilde{p} = S_0 \hat{\sigma}_1 \hat{\sigma}_2 \ldots \hat{\sigma}_k$ entnommen und wurden somit durch die Folge der Selektionsoperationen $S_0, \hat{\sigma}_1, \ldots, \hat{\sigma}_k$ aus der Gesamtmenge gewonnen.

Das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' übergibt dem Teilsystem 'WÖRTERBUCH' die aus relevanzbewerteten Dokumenten extrahierten Wortformen. Wie im Abschnitt 5.3.3.1.1 beschrieben wurde, bietet das Wörterbuch dem Benutzer Wortformen zur Präzisierung der Begriffskonstrukte an. Damit modifiziert der Benutzer jedoch die intensionale Beschreibung der von ihm verwendeten Begriffe. Das stellt sowohl die bereits getroffene Auswahl von Selektionsoperationen als auch die Extension der bisher gebildeten Objektmengen in Frage. In der Begriffswelt der Lernmodelle entspricht das der Situation, dass sich die Lernobjekte im Verlauf der Lernphase in ihren Merkmalen verändern. In einer solchen Situation muss die Lernphase noch einmal von vorn begonnen werden. Es ist zu prüfen, ob die zuvor getroffenen Entscheidungen „noch haltbar“ sind.

5.5.2.2 Der Gesamtprozess zur Konstruktion des Klassifikators

In diesem Abschnitt wollen wir beschreiben, wie im Zuge der rekursiven Prozessführung die zweite Stufe des Klassifikators aufgebaut wird, der gemeinsam mit der ersten Stufe die intensionale Beschreibung des Themas darstellt. Die zweite Stufe basiert auf der Objektmenge $\mathcal{F}_{S_0}^{4}$, die wir deshalb als ***Start-Objektmenge*** bezeichnen.

[1] Beispielsweise in Unger/Wysotzki (1981).

Die zweite Stufe des Klassifikators ist eine Regel, nach der für jedes in der Start-Objektmenge $\mathcal{F}_{S_0}^{4}$ referierte Dokument entschieden wird, ob es der Klasse „ausgeben“ oder der Klasse „nicht ausgeben“ zuzuordnen ist.

Allgemein klassifiziert ein Klassifikator $K_{\tilde{p}}^{\nu}$ sämtliche Dokumente, die in der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ referiert werden. Dabei sind zwei Situationen zu unterscheiden:

1. Der Klassifikator ordnet sämtliche in $\mathcal{F}_{\tilde{p}}^{\nu}$ referierten Dokumente derselben Klasse x zu. Wir schreiben diesen Sachverhalt in der Form:

$$K_{\tilde{p}}^{\nu} = x; \quad x \in \{+, -\} \tag{83}$$

   Dabei steht das Symbol „ + “ für die Klasse „ausgeben“ und das Symbol „ - “ für die Klasse „nicht ausgeben“.

2. Der Klassifikator enthält die Anweisung zur Zerlegung von $\mathcal{F}_{\tilde{p}}^{\nu}$ mit Hilfe einer Selektionsoperation $\sigma = (\xi, r, \nu')$. Er delegiert die Klassifizierung der in der selektierten Objektmenge $\mathcal{F}_{\tilde{p}\sigma}^{\nu'}$ referierten Dokumente an den Klassifikator $K_{\tilde{p}\sigma}^{\nu'}$ und die Klassifizierung der in der Rest-Objektmenge $\mathcal{F}_{\tilde{p}\overline{\sigma}}^{\nu}$ referierten Dokumente an den Klassifikator $K_{\tilde{p}\overline{\sigma}}^{\nu}$. Wir schreiben diesen Sachverhalt in der Form:

$$K_{\tilde{p}}^{\nu} = \left(\sigma, K_{\tilde{p}\sigma}^{\nu'}, K_{\tilde{p}\bar{\sigma}}^{\nu}\right) \tag{84}$$

Allgemein ergibt sich damit für den Klassifikator $K_{\tilde{p}}^{\nu}$ die folgende rekursive Bildungsvorschrift:

$$K_{\tilde{p}}^{\nu} = \begin{cases} x \in \{+,-\} & \mathcal{F}_{\tilde{p}}^{\nu} \text{ wird nicht zerlegt} \\ \left(\sigma, K_{\tilde{p}\sigma}^{\nu'}, K_{\tilde{p}\bar{\sigma}}^{\nu}\right) & \mathcal{F}_{\tilde{p}}^{\nu} \text{ wird mit Hilfe von} \\ & \sigma = (\xi, r, \nu') \text{ zerlegt} \end{cases} \tag{85}$$

Der Klassifikator $K_{\tilde{p}}^{\nu}$ wird im Zuge der Behandlung der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ gewonnen. Die gesuchte zweite Stufe des Klassifikators zum Thema des Benutzers entsteht durch die Behandlung der Start-Objektmenge $\mathcal{F}_{S_0}^{4}$. Wir bezeichnen die zweite Stufe des Klassifikators deshalb mit $K_{S_0}^{4}$.

5.5.2.3 Optimierung des Klassifikators

Aus der Beschreibung eines allgemeinen Rekursions-Schrittes zur Behandlung einer Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$ geht hervor, dass die Auswahl der zweckmäßigsten Selektionsoperation ohne Kenntnis derjenigen Selektionsoperationen erfolgt, die erst in ***späteren*** Rekursions-Schritten gewählt werden. Dadurch kann es vorkommen, dass in den Klassifikator $K_{\tilde{p}}^{\nu}$ ***unnötige*** Selektionsoperationen aufgenommen werden.

Das ist dann der Fall, wenn die folgende Situation vorliegt:

1. Der Klassifikator $K_{\tilde{p}}^{\nu} = \left(\sigma, K_{\tilde{p}\,\sigma}^{\nu'}, K_{\tilde{p}\,\overline{\sigma}}^{\nu}\right)$ delegiert die Klassenentscheidung - nach unter Umständen mehreren Rekursions-Schritten - letztlich auf zwei Klassifikatoren $K_{\tilde{q}_1}^{\nu_1}$ und $K_{\tilde{q}_2}^{\nu_2}$, die beide für dieselbe Klasse x entscheiden:

$$K_{\tilde{q}_1}^{\nu_1} = K_{\tilde{q}_2}^{\nu_2} = x; \qquad x \in \{+, -\} \tag{86}$$

2. Die Folgen von Selektionsoperationen $\tilde{q}_1$ und $\tilde{q}_2$ stimmen sowohl in einem Präfix $\tilde{p}$ als auch in einem Suffix $\tilde{u}$ überein und haben die Form

$$\tilde{q}_1 = \tilde{p}\,\sigma\,\tilde{u} \quad \text{und} \quad \tilde{q}_2 = \tilde{p}\,\overline{\sigma}\,\tilde{u} \tag{87}$$

Die Dokumente, die entweder durch $K_{\tilde{q}_1}^{\nu_1}$ oder durch $K_{\tilde{q}_2}^{\nu_2}$ der Klasse x zugeordnet werden, unterscheiden sich also nur dadurch, dass sie im ersten Fall durch die Selektionsoperation σ ***selektiert*** wurden und im zweiten Fall durch die Selektionsoperation σ ***nicht selektiert*** wurden. Da sie aber in beiden Fällen der Klasse x zugeordnet werden, ist für sie die Selektionsoperation σ überflüssig. Die Abbildung 21 zeigt ein Beispiel für diese Situation.

Das Auftreten unnötiger Selektionsoperationen in Klassifikatoren wurde schon in den siebziger Jahren allgemein untersucht.[1] Bei den Klassifikatoren, die nach dem hier verwendeten Verfahren entstehen, liegen jedoch spezielle Verhältnisse vor: Bei der Ausführung einer Selektionsoperation kann sich das Detailliertheitsniveau der Fundortbeschreibung in der selektierten Objektmenge verringern. Deshalb sind Selektionsoperationen in ihrer Reihenfolge nicht mehr bedingungs-

[1] Vgl. beispielsweise Martelli/Montanari (1978), Unger/Wysotzki (1981), Masahiro (1985) sowie Utgoff/Berkman/Clouse (1997).

los vertauschbar. Auch in diesem Fall lässt sich jedoch unter Umständen der Klassifikator durch das Eliminieren unnötiger Selektionsoperationen optimieren.

Wir betrachten einen Klassifikator $K_{\tilde{p}}^{\nu}$ der Form

$$K_{\tilde{p}}^{\nu} = \left(\sigma^*, K_{\tilde{p}\,\sigma^*}^{\nu^*}, K_{\tilde{p}\,\overline{\sigma}^*}^{\nu} \right); \quad \sigma^* = \left(\xi^*, r^*, \nu^* \right) \tag{88}$$

Die Selektionsoperation σ^* ist völlig (bzw. teilweise) unnötig, wenn unabhängig vom Ergebnis dieser Selektionsoperation alle (bzw. einige) Dokumente, die in $\mathcal{F}_{\tilde{p}}^{\nu}$ referiert werden, durch die beiden Teil-Klassifikatoren $K_{\tilde{p}\,\sigma^*}^{\nu^*}$ und $K_{\tilde{p}\,\overline{\sigma}^*}^{\nu}$ derselben Klasse x zugeordnet werden.

Wir untersuchen, welche Folgen es hätte, wenn die Selektionsoperation σ^* nicht ausgeführt werden würde. Wir bezeichnen den aus $K_{\tilde{p}}^{\nu}$ auf diese Weise gewonnenen Klassifikator mit $\hat{K}_{\tilde{p}}^{\nu}$ und ermitteln mit $E(\tilde{p})$ die Anzahl der dabei eingesparten Selektionsoperationen.

Wird in $K_{\tilde{p}}^{\nu} = \left(\sigma^*, K_{\tilde{p}\,\sigma^*}^{\nu^*}, K_{\tilde{p}\,\overline{\sigma}^*}^{\nu} \right)$ die Selektionsoperation σ^* nicht ausgeführt, so muss $K_{\tilde{p}}^{\nu}$ durch einen Klassifikator $\hat{K}_{\tilde{p}}^{\nu}$ ersetzt werden, der die beiden Teil-Klassifikatoren $K_{\tilde{p}\,\sigma^*}^{\nu^*}$ und $K_{\tilde{p}\,\overline{\sigma}^*}^{\nu}$ in sich vereinigt. Wir nennen das eine ***Überlagerung*** von Klassifikatoren und drücken diesen Vorgang durch die Funktion $\ddot{U}_{\nu}$ aus. Der Index ν bringt dabei zum Ausdruck, dass die

Objektmenge, für die der neu zu konstruierende Klassifikator $\hat{K}^{\nu}_{\tilde{p}}$ aufgebaut werden soll, Fundorte auf dem Niveau ν enthält:

$$\begin{aligned} \hat{K}^{\nu}_{\tilde{p}} &= \ddot{U}_{\nu}\left(K^{\nu^*}_{\tilde{p}\,\sigma^*}, K^{\nu}_{\tilde{p}\,\bar{\sigma}^*} \right) \\ E(\tilde{p}) &= 1 + G\left(K^{\nu^*}_{\tilde{p}\,\sigma^*}, K^{\nu}_{\tilde{p}\,\bar{\sigma}^*} \right) \end{aligned} \tag{89}$$

Die Einsparung $E(\tilde{p})$ ergibt sich einerseits daraus, dass mit Wegfall von σ^* ***eine*** Selektionsoperation eingespart wird. Andererseits wird durch die Funktion G der Gewinn berücksichtigt, der durch die Überlagerung der Teil-Klassifikatoren entsteht. Der Wert der Funktion G kann positiv, Null oder negativ werden.

Wir untersuchen die Überlagerung zweier Klassifikatoren $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$. Wir verzichten dabei auf die Spezifizierung der Klassifikatoren hinsichtlich des Niveaus der Fundorte, für das sie ursprünglich aufgebaut wurden. Das ist sinnvoll, weil sich durch den Wegfall der Selektionsoperation σ^* die zu klassifizierenden Fundorte ohnehin auf einem anderen - nämlich einem präziseren - Niveau befinden können.

Bei der Überlagerung der beiden Klassifikatoren $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ durch die Funktion $\ddot{U}_{\nu}$ unterscheiden wir drei Fälle:

1. $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ enthalten beide die Entscheidung für die Klasse x. Das Ergebnis der Überlagerung ist dann diese gemeinsame Klassenentscheidung. Dabei tritt natürlich ***kein Gewinn*** ein.

2. Die beiden Klassifikatoren haben die Form

$$\begin{aligned} K_{\tilde{q}_1} &= \left(\sigma, K_{\tilde{q}_1\sigma}, K_{\tilde{q}_1\overline{\sigma}}\right) \\ K_{\tilde{q}_2} &= \left(\sigma, K_{\tilde{q}_2\sigma}, K_{\tilde{q}_2\overline{\sigma}}\right) \\ \sigma &= (\xi, r, \nu') \end{aligned} \tag{90}$$

In beiden Klassifikatoren soll also dieselbe Selektionsoperation σ ausgeführt werden. Bei der Überlagerung der Klassifikatoren wird σ nur einmal übernommen. Dadurch wird ***eine*** Selektionsoperation eingespart. Die bisher vier Teil-Klassifikatoren müssen auf zwei reduziert werden: Als Klassifikator für die durch σ selektierte Objektmenge wird die Überlagerung $\ddot{U}_{\nu'}\left(K_{\tilde{q}_1\sigma}, K_{\tilde{q}_2\sigma}\right)$ verwendet. Dabei wird berücksichtigt, dass die durch $\sigma = (\xi, r, \nu')$ selektierte Objektmenge Fundorte auf dem Niveau ν' enthält. Der Klassifikator für die Rest-Objektmenge ergibt sich als Überlagerung $\ddot{U}_{\nu}\left(K_{\tilde{q}_1\overline{\sigma}}, K_{\tilde{q}_2\overline{\sigma}}\right)$. Der Gewinn beträgt also ***eine*** Selektionsoperation zuzüglich der Gewinne, die durch die beiden Überlagerungen eintreten.

3. In allen anderen Fällen ist eine Überlagerung von $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ durch die Funktion $\ddot{U}_{\nu}$ nicht möglich. Es sei

$$\begin{aligned} K_{\tilde{q}_1} &= \left(\sigma_1, K_{\tilde{q}_1\sigma_1}, K_{\tilde{q}_1\overline{\sigma}_1}\right) \\ K_{\tilde{q}_2} &= \left(\sigma_2, K_{\tilde{q}_2\sigma_2}, K_{\tilde{q}_2\overline{\sigma}_2}\right) \\ \sigma_1 &\neq \sigma_2 \end{aligned} \tag{91}$$

Die zunächst als unnötig angenommene Differenzierung der Fundorte durch die Selektionsoperation σ^* gemäß (88) erweist sich nun doch als notwendig und muss wieder eingeführt werden. Erst dann kann einerseits die durch σ^* selektierte Objektmenge durch $K_{\tilde{q}_1}$ und andererseits die

Rest-Objektmenge durch $K_{\tilde{q}_2}$ klassifiziert werden. Die Anwendung der Selektionsoperation $\sigma^* = (\xi^*, r^*, \nu^*)$ ist an dieser Stelle jedoch nur dann möglich, wenn die zu klassifizierenden Fundorte in ihrem Detailliertheitsgrad noch nicht unter dem Niveau ν^* liegen. Wir unterscheiden deshalb zwei Unterfälle:

3.1. $\nu \geq \nu^*$: Auf die zu klassifizierenden Fundorte ist die Selektionsoperation σ^* anwendbar. Der durch $\ddot{U}_\nu(K_{\tilde{q}_1}, K_{\tilde{q}_2})$ gebildete Klassifikator fügt die Selektionsoperation σ^* wieder ein und verwendet $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ als Teil-Klassifikatoren. Es muss also wieder ***eine*** Selektionsoperation hinzugefügt werden. Der Gewinn ist somit -1.

3.2. $\nu < \nu^*$: Die durch $\ddot{U}_\nu(K_{\tilde{q}_1}, K_{\tilde{q}_2})$ zu klassifizierenden Fundorte liegen bereits auf einem Niveau, das die Anwendung von σ^* nicht mehr zulässt. Die Konstruktion des optimierten Klassifikators muss dann erfolglos ***abgebrochen*** werden.

Im folgenden wird zusammenfassend die rekursive Bildungsvorschrift für die Überlagerung der Klassifikatoren $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ angegeben. Diese hängt von den Bedingungen B1, B2 und B3 ab:

B1: $K_{\tilde{q}_1} = K_{\tilde{q}_2} = x$

B2: $K_{\tilde{q}_1} = (\sigma, K_{\tilde{q}_1\sigma}, K_{\tilde{q}_1\bar{\sigma}})$ und $K_{\tilde{q}_2} = (\sigma, K_{\tilde{q}_2\sigma}, K_{\tilde{q}_2\bar{\sigma}})$

B3: $K_{\tilde{q}_1} = (\sigma_1, K_{\tilde{q}_1\sigma_1}, K_{\tilde{q}_1\bar{\sigma}_1})$ und $K_{\tilde{q}_2} = (\sigma_2, K_{\tilde{q}_2\sigma_2}, K_{\tilde{q}_2\bar{\sigma}_2})$
mit $\sigma_1 \neq \sigma_2$ und $\nu \geq \nu^*$

Die Funktion $\ddot{U}_\nu\left(K_{\tilde{q}_1}, K_{\tilde{q}_2}\right)$ wird wie folgt gebildet:

$$\ddot{U}_\nu\left(K_{\tilde{q}_1}, K_{\tilde{q}_2}\right) = \begin{cases} x & \text{B1} \\ \left(\sigma, \ddot{U}_\nu\left(K_{\tilde{q}_1\sigma}, K_{\tilde{q}_2\sigma}\right), \ddot{U}_\nu\left(K_{\tilde{q}_1\bar{\sigma}}, K_{\tilde{q}_2\bar{\sigma}}\right)\right) & \text{B2} \\ \left(\sigma^*, K_{\tilde{q}_1}, K_{\tilde{q}_2}\right) & \text{B3} \\ \text{nicht berechenbar} & \text{sonst} \end{cases} \tag{92}$$

Für den Gewinn, der durch die Überlagerung der beiden Klassifikatoren $K_{\tilde{q}_1}$ und $K_{\tilde{q}_2}$ eintritt, gilt:

$$G\left(K_{\tilde{q}_1}, K_{\tilde{q}_2}\right) = \begin{cases} 0 & \text{B1} \\ 1 + G\left(K_{\tilde{q}_1\sigma}, K_{\tilde{q}_2\sigma}\right) + G\left(K_{\tilde{q}_1\bar{\sigma}}, K_{\tilde{q}_2\bar{\sigma}}\right) & \text{B2} \\ -1 & \text{B3} \\ -\infty & \text{sonst} \end{cases} \tag{93}$$

Ein Klassifikator $K^{\nu}_{\tilde{p}} = \left(\sigma^*, K^{\nu^*}_{\tilde{p}\sigma^*}, K^{\nu}_{\tilde{p}\bar{\sigma}^*}\right)$ ist dann optimierbar, wenn $E(\tilde{p}) > 0$ ist, wenn also wenigstens eine Selektionsoperation bei der Ersetzung von $K^{\nu}_{\tilde{p}}$ durch $\hat{K}^{\nu}_{\tilde{p}} = \ddot{U}_\nu\left(K^{\nu^*}_{\tilde{p}\sigma^*}, K^{\nu}_{\tilde{p}\bar{\sigma}^*}\right)$ eingespart wird.

Wir illustrieren die Optimierung des Klassifikators an einem einfachen Beispiel. Die erlernte zweite Stufe des Klassifikators $K^4_{S_0}$ habe die folgende Form:

$$K^{4}_{S_0} = \left(\sigma_1, K^{2}_{S_0\sigma_1}, K^{4}_{S_0\overline{\sigma}_1}\right) \text{ mit } \sigma_1 = \left(\xi_1, r_1, 2\right)$$

$$K^{2}_{S_0\sigma_1} = \left(\sigma_2, K^{1}_{S_0\sigma_1\sigma_2}, K^{2}_{S_0\sigma_1\overline{\sigma}_2}\right) \text{ mit } \sigma_2 = (\xi_2, r_2, 1)$$

$$K^{1}_{S_0\sigma_1\sigma_2} = +$$

$$K^{2}_{S_0\sigma_1\overline{\sigma}_2} = -$$

$$K^{4}_{S_0\overline{\sigma}_1} = \left(\sigma_3, K^{2}_{S_0\overline{\sigma}_1\sigma_3}, K^{4}_{S_0\overline{\sigma}_1\overline{\sigma}_3}\right) \text{ mit } \sigma_3 = (\xi_3, r_3, 2)$$

$$K^{2}_{S_0\overline{\sigma}_1\sigma_3} = \left(\sigma_4, K^{2}_{S_0\overline{\sigma}_1\sigma_3\sigma_4}, K^{2}_{S_0\overline{\sigma}_1\sigma_3\overline{\sigma}_4}\right) \text{ mit } \sigma_4 = (\xi_4, r_4, 2)$$

$$K^{2}_{S_0\overline{\sigma}_1\sigma_3\sigma_4} = \left(\sigma_5, K^{2}_{S_0\overline{\sigma}_1\sigma_3\sigma_4\sigma_5}, K^{2}_{S_0\overline{\sigma}_1\sigma_3\sigma_4\overline{\sigma}_5}\right) \text{ mit } \sigma_5 = (\xi_5, r_5, 2)$$

$$K^{2}_{S_0\overline{\sigma}_1\sigma_3\sigma_4\sigma_5} = +$$

$$K^{2}_{S_0\overline{\sigma}_1\sigma_3\sigma_4\overline{\sigma}_5} = -$$

$$K^{4}_{S_0\overline{\sigma}_1\overline{\sigma}_3} = \left(\sigma_4, K^{2}_{S_0\overline{\sigma}_1\overline{\sigma}_3\sigma_4}, K^{4}_{S_0\overline{\sigma}_1\overline{\sigma}_3\overline{\sigma}_4}\right) \text{ mit } \sigma_4 = (\xi_4, r_4, 2)$$

$$K^{2}_{S_0\overline{\sigma}_1\overline{\sigma}_3\sigma_4} = \left(\sigma_5, K^{2}_{S_0\overline{\sigma}_1\overline{\sigma}_3\sigma_4\sigma_5}, K^{2}_{S_0\overline{\sigma}_1\overline{\sigma}_3\sigma_4\overline{\sigma}_5}\right) \text{ mit } \sigma_5 = (\xi_5, r_5, 2)$$

$$K^{2}_{S_0\overline{\sigma}_1\overline{\sigma}_3\sigma_4\sigma_5} = +$$

$$K^{2}_{S_0\overline{\sigma}_1\overline{\sigma}_3\sigma_4\overline{\sigma}_5} = -$$

$$K^{4}_{S_0\overline{\sigma}_1\overline{\sigma}_3\overline{\sigma}_4} = -$$

Der Klassifikator ist in Abbildung 21 in Form eines binären Baums dargestellt.

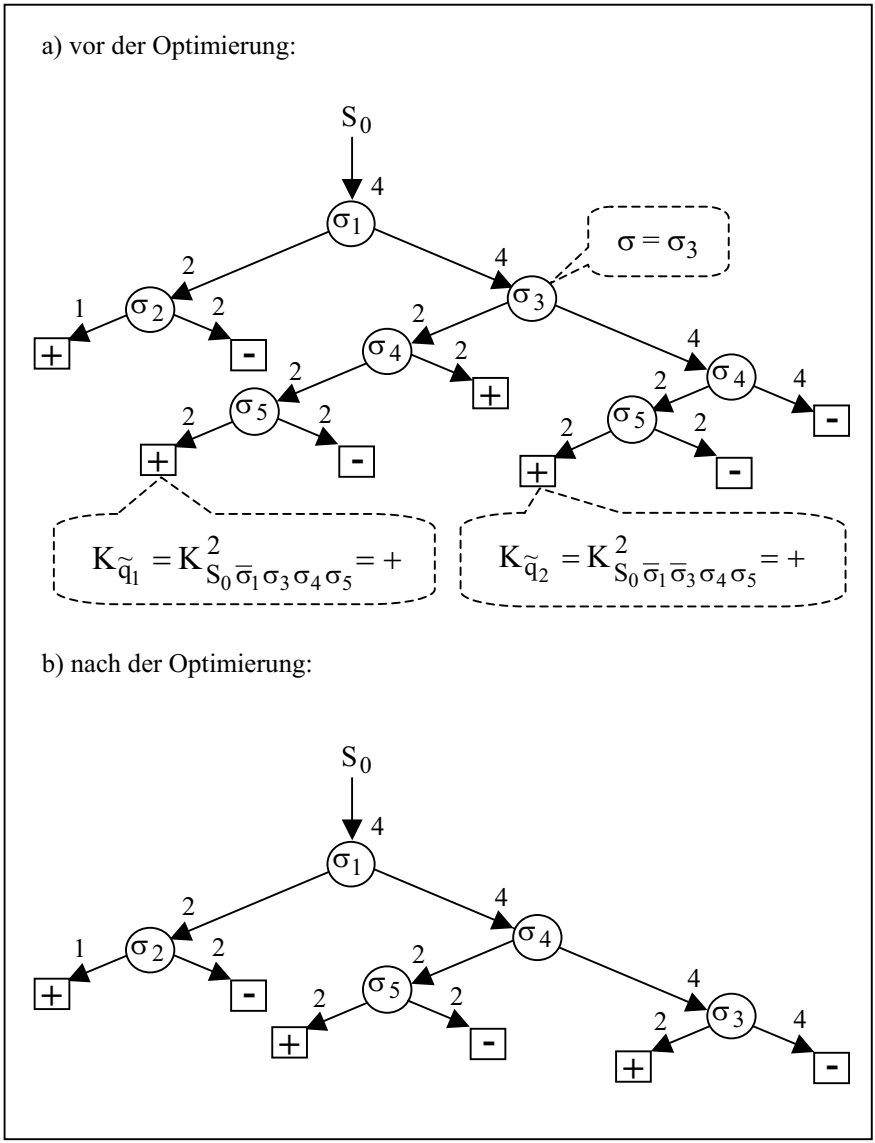


Abbildung 21: Beispielklassifikator vor (a) und nach (b) der Optimierung

Jeder Knoten des Baums entspricht einem Klassifikator $K_{\tilde{p}}$. Er enthält entweder die Entscheidung für die Klasse $x \in \{+,-\}$ (dargestellt als Quadrat) oder die Selektionsoperation σ_i (dargestellt als Kreis). Die aus der Selektionsoperation σ_i auslaufende linke Kante entspricht dem Teil-Klassifikator $K_{\tilde{p}\sigma_i}$, die auslaufende rechte Kante dem Teil-Klassifikator $K_{\tilde{p}\bar{\sigma}_i}$. An den einlaufenden Kanten ist jeweils das Niveau der Fundorte angegeben.

Die Klassifikatoren $K_{\tilde{q}_1} = K^2_{S_0\bar{\sigma}_1\sigma_3\sigma_4\sigma_5} = +$ und $K_{\tilde{q}_2} = K^2_{S_0\bar{\sigma}_1\bar{\sigma}_3\sigma_4\sigma_5} = +$ genügen den Bedingungen dafür, dass eine unnötige Selektionsoperation vorliegt:

1. $K_{\tilde{q}_1} = K_{\tilde{q}_2} = +$

2. $\tilde{q}_1 = \tilde{p}\,\sigma\,\tilde{u}$ und $\tilde{q}_2 = \tilde{p}\,\bar{\sigma}\,\tilde{u}$ mit $\tilde{p} = S_0\bar{\sigma}_1,\ \sigma = \sigma_3,\ \tilde{u} = \sigma_4\sigma_5$

Damit ist die Selektionsoperation σ_3 in den Selektionsfolgen $\tilde{q}_1$ und $\tilde{q}_2$ überflüssig und wird vorerst entfernt. Bei der Überlagerung der Klassifikatoren $K_{S_0\bar{\sigma}_1\sigma_3\bar{\sigma}_4} = +$ und $K_{S_0\bar{\sigma}_1\bar{\sigma}_3\bar{\sigma}_4} = -$ muss sie jedoch wieder eingeführt werden. Durch die Optimierung entsteht ein Klassifikator, der mit zwei Selektionsoperationen weniger auskommt.

5.5.3 Entwurf des Teilsystems 'SUCHANFRAGE KONSTRUIEREN'

Nachdem wir aus der Problemstellung die Methode entwickelt haben, nach der das Teilsystem 'SUCHANFRAGE KONSTRUIEREN' arbeiten soll, wollen wir im folgenden seine Architektur entwerfen.

Die Aufgabe des Teilsystems 'SUCHANFRAGE KONSTRUIEREN' besteht darin, einen Klassifikator zum Thema des Benutzers aufzubauen. Der Klassifikator soll für jedes Dokument der Datenbank entscheiden, ob es der Klasse „ausgeben“ oder der Klasse „nicht ausgeben“ zuzuordnen ist. Gemäß dem methodischen Konzept besteht der Klassifikator aus zwei Stufen: aus der

Grobanfrage S_0 und aus einem hierarchisch organisierten System von Teil-Klassifikatoren. Dieses System wird in Form eines binären Entscheidungsbaums[1] dargestellt.

Der Informationsverarbeitungsprozess zum Aufbau des Gesamt-Klassifikators wird durch das Zusammenspiel aller Maschinen realisiert, die im Teilsystem 'SUCHANFRAGE KONSTRUIEREN' zusammengefasst sind. Seine Struktur und seine Kommunikationsflüsse sind in der Abbildung 22 dargestellt.

Im Zeitablauf werden die folgenden Aktivitäten ausgeführt:

1. Der Aufbau des Teilsystems 'SUCHANFRAGE KONSTRUIEREN' beginnt mit der Aktivierung der Maschine 'KONSTRUKTEUR', die den ganzen weiteren Prozessablauf steuert.

2. Der 'KONSTRUKTEUR' generiert zunächst eine Maschine 'BAUM-SPEICHERQUELLE', die den Speicherplatz für den Entscheidungsbaum verwalten soll.

3. Die Maschine 'KONSTRUKTEUR' überträgt daraufhin dem Teilsystem 'KONSTRUKT RECHERCHIEREN' die Aufgabe, die Start-Objektmenge $\mathcal{F}_{S_0}^4$ zu bilden.

4. Das Teilsystem 'KONSTRUKT RECHERCHIEREN' generiert die Grobanfrage S_0 und lässt sie durch ein SEARCH-Kommando von der Maschine 'IRS' ausführen.

5. Der 'KONSTRUKTEUR' veranlasst den Aufbau des Teilsystems 'KLASSIFIKATOR AUFBAUEN' und übergibt ihm die Objektmenge $\mathcal{F}_{S_0}^4$. Die Aufgabe dieses Teilsystems, auf dessen innere Struktur wir später eingehen werden, besteht darin, auf der Grundlage der Objektmenge $\mathcal{F}_{S_0}^4$ die zweite Stufe des Klassifikators zu konstruieren.

[1] Vgl. beispielsweise Utgoff (1989), Crawford/Fung/Appelbaum/Tong (1991) sowie Blockeel/De Raedt (1998).

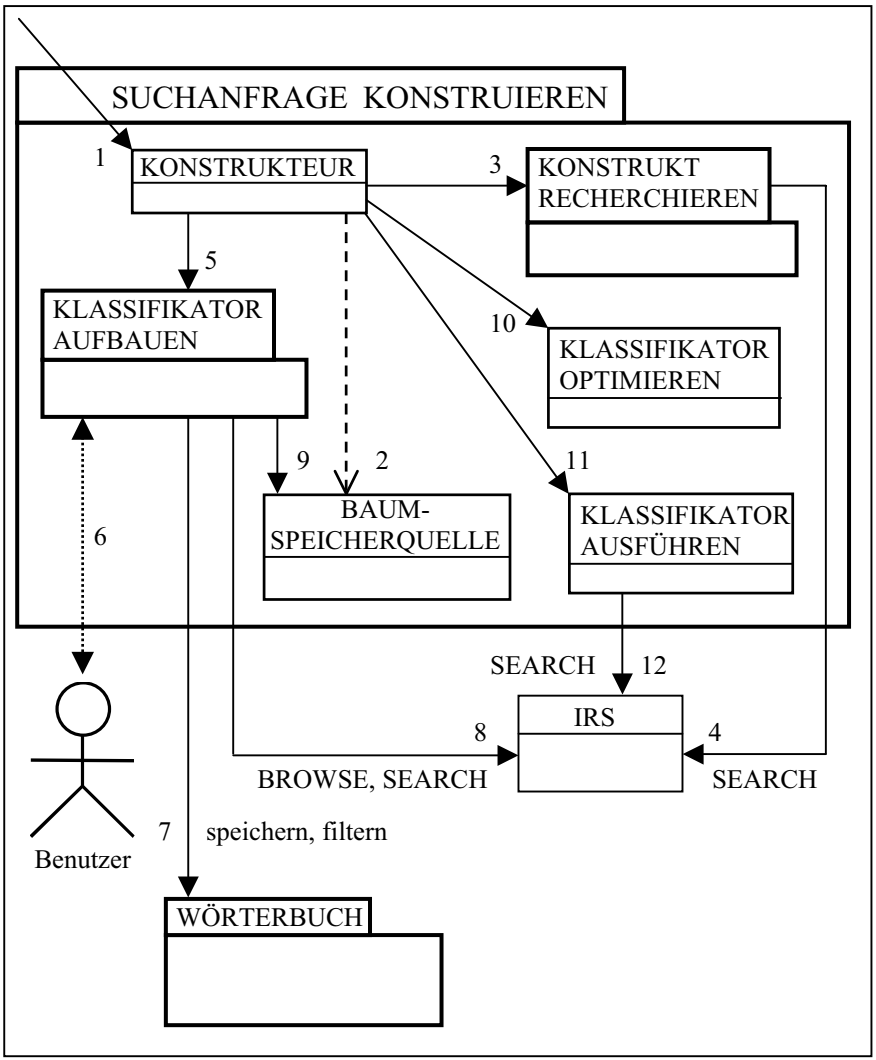


Abbildung 22: Struktur und Kommunikationsflüsse des Teilsystems 'SUCHANFRAGE KONSTRUIEREN'

6. Der Aufbau des Klassifikators erfolgt als ein überwachter Lernprozess in enger Zusammenarbeit mit dem Benutzer.

7. Im überwachten Lernprozess müssen neue Wortformen in das Teilsystem 'WÖRTERBUCH' eingespeichert werden, damit dieses befähigt wird, Offerten für den Benutzer auszufiltern.

8. Für die Konstruktion des Klassifikators ist eine Kommunikation mit der Maschine 'IRS' erforderlich. Das Information-Retrieval-System stellt einerseits in einem BROWSE-Prozess Dokumente zur Relevanzbewertung bereit. Andererseits muss die Start-Objektmenge $\mathcal{F}^4_{S_0}$ sukzessive in Teilmengen zerlegt werden. Diese Zerlegung realisiert die Maschine 'IRS' als Reaktion auf SEARCH-Kommandos.

9. Die benötigten Speicherbereiche für die Darstellung des Entscheidungsbaums fordert das Teilsystem 'KLASSIFIKATOR AUFBAUEN' von der Maschine 'BAUMSPEICHERQUELLE' an.

10. Wenn der Entscheidungsbaum vollständig aufgebaut ist, wird er von der Maschine 'KONSTRUKTEUR' zur Optimierung an die Maschine 'KLASSIFIKATOR OPTIMIEREN' übergeben.

11. Schließlich überträgt die Maschine 'KONSTRUKTEUR' der Maschine 'KLASSIFIKATOR AUSFÜHREN' die Aufgabe, den Gesamt-Klassifikator, der sich aus der Grobanfrage und aus dem optimierten Entscheidungsbaum zusammensetzt, auf die Anfragesprache des Information-Retrieval-Systems STAIRS abzubilden und durch die Maschine 'IRS' als Reaktion auf ein SEARCH-Kommando ausführen zu lassen.

12. Die Maschine 'KLASSIFIKATOR AUSFÜHREN' schickt die Suchanfrage in Form eines SEARCH-Kommando an die Maschine 'IRS'.

Wir wollen nun für das Teilsystem 'KLASSIFIKATOR AUFBAUEN' eine geeignete innere Struktur entwerfen. Diese ist gemeinsam mit den Kommunikationsflüssen in der Abbildung 23 dargestellt. Im Interesse einer besseren Übersichtlichkeit wurde auf die Generierungspfeile verzichtet.

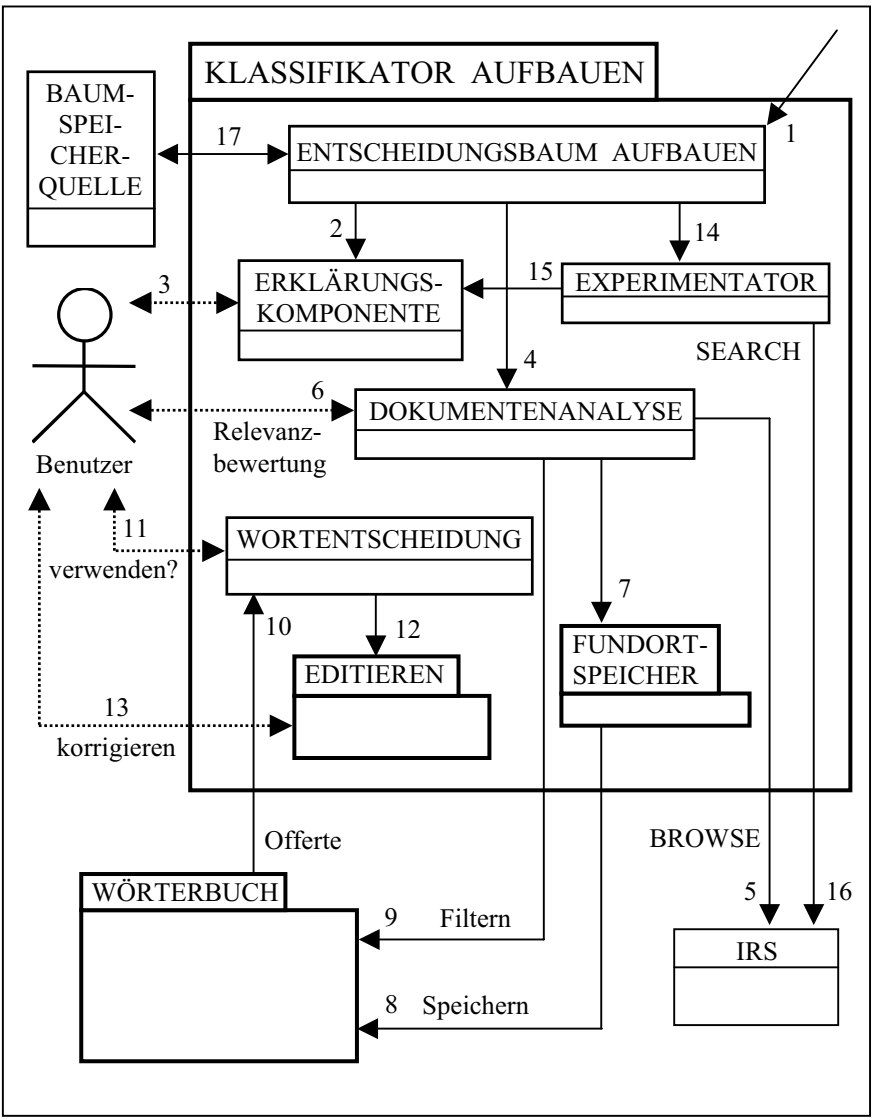


Abbildung 23: Struktur und Kommunikationsflüsse des Teilsystems 'KLASSIFIKATOR AUFBAUEN'

Im Zeitablauf werden die folgenden Aktivitäten ausgeführt:

1. Die Maschine 'KONSTRUKTEUR' initialisiert zunächst den Aufbau des Teilsystems 'KLASSIFIKATOR AUFBAUEN', indem sie die Steuerung der Maschine 'ENTSCHEIDUNGSBAUM AUFBAUEN' übergibt.

2. Die Maschine 'ENTSCHEIDUNGSBAUM AUFBAUEN' richtet das Teilsystem 'FUNDORTSPEICHER' ein und aktiviert eine Maschine 'ERKLÄRUNGSKOMPONENTE'.

3. Die Maschine 'ERKLÄRUNGSKOMPONENTE' übernimmt die Aufgabe, den Benutzer in jedem Rekursions-Schritt darüber zu informieren, mit Hilfe welcher Begriffskonstrukte bereits selektiert wurde, wie umfangreich die Objektmenge und die Lernmenge sind, wie die relevanten und irrelevanten Dokumente in der Lernmenge verteilt sind und welche Begriffskonstrukte für die Selektionsoperation empfohlen werden. Durch Kommandos steuert der Benutzer den weiteren Ablauf.

4. Möchte der Benutzer, dass ihm Dokumente zur Relevanzbewertung angeboten werden, dann generiert die Maschine 'ENTSCHEIDUNGSBAUM AUFBAUEN' die beiden Maschinen 'DOKUMENTENANALYSE' und 'WORTENTSCHEIDUNG'. Im Anschluss daran wird die Maschine 'DOKUMENTENANALYSE' aktiviert.

5. Die Maschine 'DOKUMENTENANALYSE' fordert unter Verwendung des BROWSE-Kommandos von der Maschine 'IRS' Dokumente ab.

6. Die Dokumente werden dem Benutzer zur Relevanzbewertung vorgelegt.

7. 'DOKUMENTENANALYSE' extrahiert aus diesen Dokumenten Wortformen und übergibt sie gemeinsam mit ihren Fundorten und der Relevanzbewertung des Benutzers an den 'FUNDORTSPEICHER'.

8. Der 'FUNDORTSPEICHER' leitet die Wortformen zur Verwaltung an das 'WÖRTERBUCH' weiter.

9. Am Ende jedes Dokuments wird das 'WÖRTERBUCH' aufgefordert, Wortformen auszufiltern.

10. Die ausgefilterten Wortformen werden als Offerten an die Maschine 'WORTENTSCHEIDUNG' übergeben.

11. Die Maschine 'WORTENTSCHEIDUNG' fragt den Benutzer, ob er die Offerten verwenden möchte.

12. Ist das der Fall, aktiviert die Maschine 'WORTENTSCHEIDUNG' das Teilsystem 'EDITIEREN'.

13. Das Teilsystem 'EDITIEREN' unterstützt den Benutzer bei der Korrektur seiner Begriffskonstrukte.

14. Zur Ermittlung von zweckmäßigen Selektionsoperationen aktiviert die Maschine 'ENTSCHEIDUNGSBAUM AUFBAUEN' eine Maschine 'EXPERIMENTATOR'.

15. Der 'EXPERIMENTATOR' führt in der Lernmenge Zerlegungs-Experimente durch, um zweckmäßige Selektionsoperationen zu ermitteln. Die zweckmäßigsten Selektionsoperationen werden dem Benutzer durch Vermittlung der 'ERKLÄRUNGSKOMPONENTE' angeboten.

16. Hat sich der Benutzer für eine Selektionsoperation entschieden, so wird der 'EXPERIMENTATOR' damit beauftragt, diese Selektionsoperation nun tatsächlich zunächst in der Lernmenge (in Kooperation mit der Maschine 'FUNDORTSPEICHER') und dann auch in der Objektmenge (durch SEARCH-Kommandos an die Maschine 'IRS') auszuführen.

17. Die Selektionsoperation wird in den Klassifikator aufgenommen, der unter Verwendung der Maschine 'BAUMSPEICHERQUELLE' als Baumstruktur gespeichert wird.

Der Aufbau und die Optimierung der zweiten Stufe des Klassifikators wurden im Abschnitt 5.4.2 beschrieben. Offen ist jedoch noch, in welcher Weise der optimierte Gesamt-Klassifikator auf die Anfragesprache des Information-Retrieval-Systems STAIRS abgebildet und in der Datenbank ausgeführt wird.

Bei dieser Abbildung stoßen wir auf das bereits besprochene Problem, dass die in der Lernmenge vollzogene Restmengenbildung durch STAIRS nicht in gleicher Weise in der Objektmenge nachvollzogen werden kann. Wir müssen sichern, dass bei der Ausgabe der in einer Rest-Objektmenge referierten Dokumente nicht auch solche Dokumente ausgegeben werden, für die in der selektierten Objektmenge bereits entschieden wurde, dass sie nicht auszugeben sind. Deshalb bilden wir im Zuge der Abarbeitung des Klassifikators die Menge $\mathcal{N}$ derjenigen Dokumente, die nicht ausgegeben werden sollen, und korrigieren mit ihrer Hilfe jede auszugebende Objektmenge.

Dieses Vorgehen zeigt das Zusammenspiel der Algorithmen 'Suchanfrage in STAIRS ausführen' und 'Klassifikator ausführen'. Die Funktion 'Referenznummer' lässt ihr Argument durch STAIRS abarbeiten und gibt als Funktionswert die Referenznummer des Selektionsergebnisses zurück. Die Funktion 'Operator' ermittelt den anzuwendenden Operator gemäß der folgenden Tabelle:

ξ	ν'	Operator(ξ, ν')
+1	3	SENT
+1	2	SEGM
+1	1	AND
-1	3	NOTSENT
-1	2	NOTSEGM
-1	1	NOT

Algorithmus: Suchanfrage in STAIRS ausführen

Eingabe: Klassifikator $K^4_{S_0}$ als zweite Stufe der Suchanfrage

Ausgabe: Referenznummer n_R des Ergebnisses der Suchanfrage

Vorgehen:

var $n_{r_1}, n_{r_2}, \ldots, n_{r_k}$, **co** Referenznummern der Begriffskonstrukte der Menge $R' = \{r_1, r_2, \ldots, r_k\}$ **oc**
n_{S_0}, **co** Referenznummer der Grobanfrage S_0 **oc**
$n_{\mathcal{N}}$, **co** Referenznummer der Menge $\mathcal{N}$ **oc**
n_{ε} **co** Referenznummer der leeren Objektmenge **oc**

begin

n_{ε} :=Referenznummer (X NOT X); **co** leere Objektmenge **oc**
$n_R := n_{\varepsilon}$; **co** Das Selektionsergebnis ist zu Beginn leer **oc**
$n_{\mathcal{N}} := n_{\varepsilon}$; **co** Die Menge $\mathcal{N}$ ist zu Beginn leer **oc**

co Alle Begriffskonstrukte werden abgearbeitet: **oc**
for $r_i \in R'$ **do** n_{r_i} :=Referenznummer(r_i);

co Die Start-Objektmenge $\mathcal{F}^4_{S_0}$ wird durch die erste Stufe des Klassifikators (Grobanfrage) gebildet: **oc**
n_{S_0} :=Referenznummer(n_{r_1} OR n_{r_2} OR … OR n_{r_k});

co Die zweite Stufe des Klassifikators wird ausgeführt. Durch die rekursive Prozedur 'Klassifikator ausführen' wird die Referenznummer n_R (globale Variable) des Ergebnisses der Suchanfrage belegt: **oc**
Klassifikator ausführen($n_{S_0}, K^4_{S_0}$);

end

Algorithmus: Klassifikator ausführen

Eingabe: Referenznummer $n_{\tilde{p}}^{\nu}$ der Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$

Klassifikator $K_{\tilde{p}}^{\nu}$ für die Objektmenge $\mathcal{F}_{\tilde{p}}^{\nu}$

Ausgabe: Modifizierung der globalen Variablen n_R und $n_{\mathcal{N}}$

Vorgehen:

begin

if $K_{\tilde{p}}^{\nu} = +$ **then**

begin

co Das Ergebnis der Suchanfrage wird erweitert: **oc**

n_R :=Referenznummer(n_R OR ($n_{\tilde{p}}^{\nu}$ NOT $n_{\mathcal{N}}$));

end else

if $K_{\tilde{p}}^{\nu} = -$ **then**

begin

co Die Menge $\mathcal{N}$ wird erweitert: **oc**

$n_{\mathcal{N}}$:=Referenznummer($n_{\mathcal{N}}$ OR $n_{\tilde{p}}^{\nu}$);

end else

begin

co Der Klassifikator hat die Form:

$K_{\tilde{p}}^{\nu} = \left(\sigma, K_{\tilde{p}\,\sigma}^{\nu'}, K_{\tilde{p}\,\bar{\sigma}}^{\nu}\right)$ mit $\sigma = \left(\xi, r_i, \nu'\right)$

oc

Klassifikator ausführen

(Referenznummer($n_{\tilde{p}}^{\nu}$ Operator(ξ, ν') n_{r_i}), $K_{\tilde{p}\,\sigma}^{\nu'}$);

Klassifikator ausführen($n_{\tilde{p}}^{\nu}, K_{\tilde{p}\,\bar{\sigma}}^{\nu}$);

end;

end

5.6 Erreichte Ergebnisse beim Entwurf des KI-Assistenten

Am Beispiel eines KI-Assistenten, der den Benutzer bei der Online-Recherche unterstützt, wurde in diesem Kapitel gezeigt, dass man den Zyklus des Software-Entwurfs in der Programmierumgebung von KOMPROMISS zweckmäßigerweise in drei Phasen unterteilt: Zunächst werden ***theoretische Modelle*** für die Lösungsmethoden erarbeitet, aus denen dann eine zweckmäßige ***Architektur von Teilsystemen abstrakter Maschinen*** abgeleitet wird. Schließlich werden die ***Algorithmen*** erarbeitet, nach denen die abstrakten Maschinen arbeiten sollen.

Bei der Ableitung einer zweckmäßigen Architektur für ein Teilsystem abstrakter Maschinen zeigte es sich, dass KOMPROMISS die Entwurfsmethode des „stepwise refinement“ gut unterstützt.

Die Maschinen 'WORTSPEICHERQUELLE' und 'WORTLISTE' im Teilsystem 'WÖRTERBUCH' sind Beispiele für abstrakte Datentypen. Das spezielle Zusammenspiel der beiden Maschinen 'DOKUMENTENANALYSE' und 'WORTENTSCHEIDUNG' im Teilsystem 'BEGRIFFKONSTRUKTE PRÄZISIEREN' zeigt, dass durch KOMPROMISS auch Koprogramme realisiert werden können. Am Beispiel der Algorithmen zur Manipulation von Präfixbäumen und Entscheidungsbäumen wurde die rekursive Programmierung in der Programmiersprache KOMPROMISS demonstriert.

Das qualitativ höhere Niveau, auf das die Mensch-Maschine-Kommunikation durch den KI-Assistenten angehoben wird, äußert sich darin, dass der Benutzer den Dialog mit dem Information-Retrieval-System nicht mehr auf der Kommandoebene des IRS, sondern auf der Ebene einer intelligenten Benutzeroberfläche führt.

Die durch die Anwenderprogramme realisierte zusätzliche Intelligenz basiert auf dem programmierten Dialog des KI-Assistenten mit dem IRS. Die Auswertung der dabei vom IRS bereitgestellten Informationen erfolgt durch Algorithmen der Massendatenverarbeitung. Ein Beispiel dafür ist der Filterprozess, der von der Maschine 'HYPOTHESENPRÜFUNG' im Teilsystem 'WÖRTERBUCH' ausgeführt wird.

Die konkrete Gestaltung der Mensch-Maschine-Kommunikation ist durch Lernprozesse geprägt, die nach Methoden des überwachten und des nichtüberwachten Lernens ablaufen. Wie dabei „Vorerfahrung“ genutzt, Entscheidungs-

regeln verwendet und konkretes „Wissen“ in den Lernprozess eingebracht wird, wurde in den Teilsystemen 'BEGRIFFSKONSTRUKTE PRÄZISIEREN' und 'SUCHANFRAGE KONSTRUIEREN' demonstriert.

6 Fazit und Ausblick

Mit der kommerziellen Etablierung großer Information-Retrieval-Systeme und der Erleichterung ihres Zugriffs über das World Wide Web wurde die Möglichkeit geschaffen, ein breites Spektrum von bibliographischen Datenbanken für einen großen Benutzerkreis verfügbar zu halten. Mit wachsender methodischer Erfahrung bei der Informationsversorgung der Benutzer durch Online-Systeme erhöht sich die Notwendigkeit, diese Erfahrungen in den praktischen Rechercheablauf einfließen zu lassen. Dabei sind zwei gegenläufige Interessen miteinander zu vereinen:

- Einerseits sollen neue Verfahren möglichst rasch in die Benutzeroberflächen der Information-Retrieval-Systeme integriert werden.

- Andererseits gefährdet jede Änderung der erprobten Software den stabilen Routinebetrieb und erfordert deshalb besondere Vorsicht.

In der Forschung wird daher nach Lösungen gesucht, durch die sich zusätzliche Intelligenz ohne Risiko und in einfacher Weise in die bestehenden Information-Retrieval-Systeme einfügen lässt.

Das vorliegende Buch will zur Lösung dieses Problems beitragen. Für das Information-Retrieval-System STAIRS - als Prototyp für eine ganze Klasse von Information-Retrieval-Systemen - wurde eine Methode entwickelt, nach der Anwender-Programme zur Realisierung zusätzlicher Intelligenz in das Information-Retrieval-System integriert werden können, ohne dass dazu Veränderungen an der bewährten Software vorgenommen werden müssen. Das Grundkonzept dieser Methode besteht darin, das Information-Retrieval-System je nach Erfordernis durch einen speziellen KI-Assistenten zu komplettieren, der die zusätzliche Intelligenz realisiert.

Um eine ausgewogene Lösung zu schaffen, erfolgte die Themenbearbeitung auf drei Ebenen: aus der theoretischen Formulierung des Lösungsverfahrens wurde eine Programmiersprache abgeleitet, die dann beim Entwurf eines speziellen KI-Assistenten erprobt wurde:

1. Die theoretische Grundlage für die Realisierung eines KI-Assistenten bildet das Modell eines ***Systems abstrakter Maschinen***. Die abstrakten Maschinen sind voneinander unabhängige informationsverarbeitende Einheiten, die ihr eigenes „Gedächtnis" haben und nur in diskreten Sendepunkten durch Nachrichtenaustausch miteinander in Beziehung treten. Dabei findet zugleich eine Steuerungsübergabe statt, so dass die quasiparallele Arbeit der Einheiten realisiert werden kann. Da es gelang, das Information-Retrieval-System STAIRS ebenfalls als eine abstrakte Maschine in das System zu integrieren, erfolgt auch die Zusammenarbeit des KI-Assistenten mit dem Information-Retrieval-System lediglich durch den Austausch von Nachrichten. So konnte das Ziel erreicht werden, die erprobte Software von STAIRS unverändert zu erhalten.

2. Für die Formulierung der Anwenderprogramme wurde die ***Programmiersprache KOMPROMISS*** entwickelt, die PL/I als Wirtssprache verwendet. Kernstück von KOMPROMISS ist ein erweitertes Steuerprogramm, das als Laufzeitsystem für die Anwenderprogramme fungiert.

3. Um zu demonstrieren, dass KOMPROMISS auch bei komplizierten Aufgaben ein praktikables Werkzeug zum Entwurf von Anwenderprogrammen darstellt, wurde ein ***KI-Assistent zur Unterstützung des Benutzers*** bei der Online-Recherche realisiert. Den modernen Prinzipien des Software-Entwurfs folgend, wurde für die Lösung der einzelnen Teilaufgaben zunächst eine theoretische Formulierung gegeben. Daraus wurde eine zweckmäßige Architektur für ein Teilsystem abstrakter Maschinen einschließlich ihrer Kommunikationsbeziehungen abgeleitet. Schließlich wurden für die abstrakten Maschinen die Algorithmen entwickelt.

Für die methodisch interessanten Teilsysteme des KI-Assistenten wurde der Entwurfszyklus in diesem Buch durchlaufen. Da die Sprache KOMPROMISS die Entwicklung von Anwender-Programmen zur Realisierung zusätzlicher Intelligenz unterstützen soll, lag bei der Auswahl der Beispiele der Schwerpunkt auf dem Gebiet lernfähiger abstrakter Maschinen. Damit wurde die Eignung von KOMPROMISS für die Programmierung sowohl von nichtüberwachten als auch

von überwachten Lernprozessen nachgewiesen. In der Kommunikation mit dem Benutzer wurde auf Erklärungs- und Schlussfolgerungskomponenten Wert gelegt. In Realisierung eines Grundprinzips der Massendatenverarbeitung wurden Filterprozesse programmiert, um unwesentliche Informationen möglichst frühzeitig von der weiteren Verarbeitung auszuschließen.

Bei der theoretischen Vertiefung und in der praktischen Erprobung des Konzepts der KI-Assistenten zeigten sich dessen Vorzüge, traten aber auch einige Schwächen hervor.

Von besonderem Nutzen erwies sich die Grundausrichtung von Theorie und Sprache auf die modulare Architektur von Maschinensystemen. Besonders wertvolle Möglichkeiten ergaben sich aus dem Prozess-Konzept. Damit ließ sich eine objektorientierte Programmierung verwirklichen, wie sie beispielsweise beim Entwurf von abstrakten Datentypen und Koprogrammen benötigt wird. Bewährt hat sich auch das beim Entwurf von KOMPROMISS verfolgte Konzept, rekursive Programmiertechniken zu unterstützen. Die Anwendungsbeispiele belegen die Vorteile, die sich aus rekursiven Steuerungsübergaben sowohl zwischen abstrakten Maschinen als auch innerhalb einer Maschine ergeben. Das in KOMPROMISS realisierte Prinzip, für häufig wiederkehrende Aufgaben - wie beispielsweise die lexikalische Analyse, die Führung von Magazinspeichern oder die Verarbeitung von Zeichenketten - vorgefertigte Werkzeuge anzubieten, ermöglicht die Formulierung kurzer und übersichtlicher Programme.

Solange der Programmierer die Daten ausschließlich im Hauptspeicher ablegt, braucht er nicht zu berücksichtigen, dass er Software für ein Teilhabersystem schreibt. Wenn im realen Betrieb mehrere Benutzer gleichzeitig mit dieser Software arbeiten, sichert KOMPROMISS, dass jeder Benutzer nur Zugriff auf seine „persönlichen“ Datenobjekte erhält.

Anders verhält es sich dagegen mit Datensammlungen auf externen Speichern. Solche Dateien gelten als gemeinschaftliche Ressourcen, auf die alle Benutzer zugreifen können. Das kann zu Konflikten führen. Die gegenwärtige Version von KOMPROMISS bietet für die Lösung solcher Konflikte keine Werkzeuge an. Eine Erweiterung von KOMPROMISS in dieser Richtung ist jedoch unproblematisch. Dazu müssen die von CICS und STAIRS bereitgestellten Mechanismen, die eine zeitweilige exklusive Nutzung von Ressourcen durch nur einen Benutzer sichern, auch in KOMPROMISS verfügbar gemacht werden.

Solche Erweiterungen wären auch nützlich, um eine Kooperation zwischen den KI-Assistenten zu ermöglichen, die für verschiedene Benutzer arbeiten. Damit ließe sich in einfacher Weise ein Nachrichtenaustausch zwischen mehreren Benutzern zur Unterstützung arbeitsteiliger Prozesse verwirklichen. Das wäre ein wichtiger Schritt in die zukunftsträchtige Richtung des ***agentenorientierten Information Retrieval***.[1]

[1] Vgl. beispielsweise Witten/Nevill-Manning/Maulsby (1996) und Klusch/Bergamaschi/Edwards/Petta (2003).

Literaturverzeichnis

Abts/Mülder (2002): Abts, D.; Mülder, W.: *Grundkurs Wirtschaftsinformatik*. 4. Auflage. Braunschweig, Wiesbaden: Friedr. Vieweg & Sohn, 2002.

Airio (2006): Airio, E.: *Word normalization and decompounding in mono- and bilingual IR*. Information Retrieval **9**(2006)1, S. 249-271.

Alkula (2001): Alkula, R.: *From Plain Character Strings to Meaningful Words: Producing Better Full Text Databases for Inflectional and Compounding Languages with Morphological Analysis Software*. Information Retrieval **4**(2001)3-4, S. 195-208.

Allan (1996): Allan, J.: *Incremental relevance feedback for information filtering*. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1996, S. 270-278.

Allan/Bolivar/Wade (2003): Allan, J.; Bolivar, A.; Wade, C.: *Retrieval and Novelty Detection at the Sentence Level*. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003, S. 314-321.

Allan/Kumaran (2003): Allan, J.; Kumaran, G.: *Stemming in the Language Modeling Framework*. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003, S. 455-456.

Anick (1999): Anick, P.: *The paraphrase search assistant: terminological feedback for iterative information seeking*. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1999, S. 153-159.

Asonov (2004): Asonov, D.: *Querying databases privatly: A new approach to private information retrieval*. Berlin, ...: Springer-Verlag, 2004.

Atkinson/Bower/Crothers (1965): Atkinson, R. C.; Bower, G. H.; Crothers, E. J.: *An Introduction to Mathematical Learning Theory.* New York, London, Sydney: John Wiley & Sons, 1965.

Baeza-Yates/Ribeiro-Neto (1999): Baeza-Yates, R.; Ribeiro-Neto, B.: *Modern information retrieval*. Harlow u.a.: Addison-Wesley, 1999.

Bai/Song/Bruza/Nie/Cao (2005): Bai, J.; Song, D.; Bruza, P.; Nie, J.-Y.; Cao, G.: *Query expansion using term relationships in language models for information retrieval*. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 2005.

Beaulieu (1997): Beaulieu, M.: *Experiments on interfaces to support query expansion*. Journal of Documentation **53**(1997)1, S. 8-19.

Beierle/Kern-Isberner (2000): Beierle, Ch.; Kern-Isberner, G.: *Methoden wissensbasierter Systeme. Grundlagen, Algorithmen, Anwendungen*. Braunschweig, Wiesbaden: Friedr. Vieweg & Sohn, 2000.

Belkin/Cool/Kelly/Lin/Park/Perez-Carballo/Sikora (2001): Belkin, N. J.; Cool, C.; Kelly, D.; Lin, S.-J.; Park, S. Y.; Perez-Carballo, J.; Sikora, C.: *Iterative exploration, design and evaluation of support for query reformulation in interactive information retrieval*. Information Processing and Management **37**(2001)3, S. 403-434.

Belkin/Cool/Stein/Thiel (1995): Belkin, N. J.; Cool, C.; Stein, A.; Thiel, U.: *Cases, Scripts, and Information-Seeking Strategies: On the Design of Interactive Information Retrieval Systems*. Expert Systems with Applications **9**(1995)3, S. 379-395.

Belkin/Marchetti/Cool (1993): Belkin, N. J.; Marchetti, P. G.; Cool, C.: *BRAQUE: Design of an interface to support user interaction in information retrieval*. Information Processing and Management **29**(1993)3, S. 325-344.

Bishop (2006): Bishop, C. M.: *Pattern Recognition and Machine Learning*. New York: Springer-Verlag, 2006.

Blockeel/De Raedt (1998): Blockeel, H.; De Raedt, L.: *Top-down induction of first order logical decision trees*. Artificial Intelligence **101**(1998)1-2, S. 285-297.

Blosseville/Hebrail/Monteil/Penot (1992): Blosseville, M. J.; Hebrail, G.; Monteil, M. G.; Penot, N.: *Automatic document classification: natural language processing, statistical analysis, and expert system techniques used together*. In: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1992, S. 51-57.

Boström (1998): Boström, H.: *Predicate invention and learning from positive examples only*. In: Proceedings of the 10th European Conference on Machine Learning, 1998.

Boyan/Freitag/Joachims (1996): Boyan, J.; Freitag, D.; Joachims, T.: *A Machine Learning Architecture for Optimizing Web Search Engines*. In AAAI Workshop on Internet Based Information Systems, 1996.

Brajnik/Guida/Tasso (1988): Brajnik, G.; Guida, G.; Tasso, C.: *IR-NLI II: Applying man-machine interaction and artificial intelligence concepts to information retrieval*. In: Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1988, S. 387-399.

Brajnik/Mizzaro/Tasso (2002): Brajnik, G.; Mizzaro, S.; Tasso, C.: *Strategic help in user interfaces for information retrieval*. Journal of the American Society for Information Science and Technology **53**(2002)5, S. 343-358.

Braschler/Ripplinger (2004): Braschler, M.; Ripplinger, B.: *How Effective is Stemming and Decompounding for German Text Retrieval?* Information Retrieval **7**(2004)3-4, S. 291-316.

Brügge/Dutoit (2004): Brügge, B.; Dutoit, A. H.: *Objektorientierte Softwaretechnik*. München: Pearson Education, 2004.

Buckley/Salton/Allan (1994): Buckley, C.; Salton, G.; Allan, J.: *The effect of adding relevance information in a relevance feedback environment*. In: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1994, S. 292-300.

Byrd/Ravin/Prager (1995): Byrd, R.; Ravin, Y.; Prager, J.: *Lexical Assistance at the Information Retrieval User Interface*. Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval. Las Vegas, Nevada, 1995.

Callan (1998): Callan, J.: *Learning While Filtering Documents*. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, S. 224-231.

Cao/Nie/Bai (2005): Cao, G.; Nie, J.-Y.; Bai, J.: *Integrating word relationships into language models*. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2005, S. 298-305.

Carbonell/Michalski/Mitchell (1983): Carbonell, J. G.; Michalski, R. S.; Mitchell, T. M.: *An overview of machine learning*. In: Michalski, R. S.; Carbonell, J. G.; Mitchell, T. M. (Hrsg): Machine Learning. An Artificial Intelligence Approach. Palo Alto, CA: Tioga Publishing Company, 1983, S. 3-23.

Carmel/Farchi/Petruschka/Soffer (2002): Carmel, D.; Farchi, E.; Petruschka, Y.; Soffer, A.: *Automatic query refinement using lexical affinities with maximal information gain*. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2002, S. 283-290.

Carpineto/de Mori/Romano/Bigi (2001): Carpineto, C.; de Mori, R.; Romano, G.; Bigi, B.: *An information-theoretic approach to automatic query expansion*. ACM Transactions on Information Systems (TOIS) **19**(2001)1, S. 1-27.

Chang/Ounis/Kim (2006): Chang, Y.; Ounis, I.; Kim, M.: *Query reformulation using automatically generated query concepts from a document space*. Information Processing and Management **42**(2006)2, S. 453-468.

Chen (1995): Chen, H.: *Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms*. Journal of the American Society for Information Science **46**(1995)3, S. 194-216.

Chen/Karger (2006): Chen, H.; Karger, D. R.: *Less is more: probabilistic models for retrieving fewer relevant documents*. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2006, S. 429-436.

Chen/Kim (1995): Chen, H.; Kim, J.: *GANNET: a machine learning approach to document retrieval*. Journal of Management Information Systems **11**(1995)3, S. 7-41.

Chowdhury (2004): Chowdhury, G. G.: *Introduction to modern information retrieval*. London: Facet, 2004.

CICS (1982): *Customer Information Control System / Virtual Storage (CICS/VS). General Information*. IBM: Order No. GC33-0155-1, 1982.

CICS (2006): *CICS Transaktion Server for OS/390 Release Guide*. IBM: Order No. GC34-5352-38, 2006.

Cohen/Singer (1996): Cohen, W. W.; Singer, Y.: *Context-sensitive learning methods for text categorization*. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1996, S. 307-315.

Cohn/Ghahramani/Jordan (1996): Cohn, D. A.; Ghahramani, Z.; Jordan, M. I.: *Active learning with statistical models*. Journal of Artificial Intelligence Research **4**(1996), S. 129-145.

Crawford/Fung/Appelbaum/Tong (1991): Crawford, S. L.; Fung, R.; Appelbaum, L. A.; Tong, R. M.: *Classification trees for information retrieval*. In: Proceedings of the 8th International Workshop on Machine Learning, 1991, S. 245-249.

Croft (1995): Croft, W. B.: *Effective Text Retrieval Based on Combining Evidence from the Corpus and Users*. IEEE Expert: Intelligent Systems and their Applications **10**(1995)6, S. 59-63.

Croft/Krovetz/Turtle (1990): Croft, W. B.; Krovetz, R.; Turtle, H.: *Interactive retrieval of complex documents*. Information Processing and Management **26**(1990), S. 593-613.

Cummins/O'Riordan (2005): Cummins, R.; O'Riordan, C.: *An evaluation of evolved term-weighting schemes in information retrieval*. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 2005.

Date (1999): Date, C. J.: *An Introduction to Database Systems*, 7th ed., Boston, MA: Addison-Wesley, 1999.

Date/Darwen (1998): *SQL – Der Standard*. Boston, MA: Addison-Wesley, 1998.

de Kroon/Mitchell/Kerckhoffs (1996): de Kroon, E.; Mitchell, T; Kerckhoffs, E.: *Improving Learning Accuracy in Information Filtering*. In: Proceedings of the International Conference on Machine Learning (ICML), Workshop on Machine Learning meets Human Computer Interaction, 1996, S. 41-58.

Dennis/McArthur/Bruza (1998): Dennis, S.; McArthur, R.; Bruza, P.: *Searching on the World Wide Web Made Easy? The Cognitive Load Imposed by Query Refinement Mechanisms*. In: Proceedings of the 20th Annual Conference of the Cognitive Science Society. Mahwah, NJ: Lawrence Erlbaum, 1998.

de Raedt (1997): de Raedt, L.: *Logical settings for concept-learning*. Artificial Intelligence **95**(1997)20, S. 187-201.

Diaz/Allan (2006): Diaz, F.; Allan, J.: *When Less is More: Relevance Feedback Falls Short and Term Expansion Succeds at HARD 2005*. Online Proceedings of 2005 Text REtrieval Conference (TREC 2005), 2006, S. 833-838.

Dumais/Platt/Heckerman/Sahami (1998): Dumais, S.; Platt, J.; Heckerman, D.; Sahami, M.: *Inductive learning algorithms and representations for text categorization*. In: Proceedings on the 7th International Conference on Information and Knowledge Management (CIKM '98), 1998, S. 148-155.

Eastman/Jansen (2003): Eastman, C. M.; Jansen, B. J.: *Coverage, Relevance, and Ranking: The Impact of Query Operators on Web Search Results*. ACM Transactions on Information Systems (TOIS) **21**(2003)4, S. 383-411.

Eguchi (2005): Eguchi, K.: *Query Expansion Experiments using Term Dependence Models*. In: Proceedings of the 5th NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access, 2005, S. 494-501.

Elmasri/Navathe (2002): Elmasri, R.; Navathe, S. B.: *Grundlagen von Datenbanksystemen*. 3. Auflage. München: Pearson Studium, 2002.

Ernst (2003): Ernst, H.: *Grundkurs Informatik*. 3. Auflage. Braunschweig, Wiesbaden: Friedr. Vieweg & Sohn, 2003.

Feddema (2002): Feddema, H.: *Microsoft Access Version 2002 Inside Out*. Redmond, WA: Microsoft Press, 2002.

Ferber (2003): Ferber, R.: *Information Retrieval: Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. Heidelberg: dpunkt-Verlag, 2003.

Ferber/Wettler/Rapp (1995): Ferber, R.; Wettler, M.; Rapp, R.: *An Associative Model of Word Selection in the Generation of Search Queries*. Journal of the American Society for Information Science **46**(1995)9, S. 685-699.

Fischer (1994): Fischer, M.: *Weiterentwicklung und Implementierung eines Dialogmodells für kooperative Informationssysteme*. Sankt Augustin: GMD, 1994.

Forsyth/Rada (1986): Forsyth, R. S.; Rada, R.: *Machine learning: Applications in expert systems and information retrieval*. New York, NY: Halsted Press, 1986.

Frakes (1992): Frakes, W. B.: *Stemming Algorithms*. In: Frakes, W. B.; Baeza-Yates, R. (Hrsg.): Information Retrieval: Data Structures and Algorithms. Upper Saddle River, NJ: Prentice Hall, 1992, S. 131-160.

Frants/Shapiro (1991): Frants, V. I.; Shapiro, J.: *Control and feedback in a documentary information retrieval system*. Journal of the American Society for Information Science **42**(1991), S. 623-634.

Frants/Shapiro/Taksa/Voiskunskii (1999): Frants, V. I.; Shapiro, J.; Taksa, I.; Voiskunskii, V. G.: *Boolean Search: Current State and Perspectives*. Journal of the American Society for Information Science **50**(1999)1, S. 86-95.

Fuhr (2004): Fuhr, N.: *Theorie des Information Retrieval I: Modelle*. In: Kuhlen, R.; Seeger, T.; Strauch, D. (Hrsg): Grundlagen der praktischen Information und Dokumentation. 5. Aufl. München: K. G. Saur, 2004, S. 207-214.

Fuhr/Pfeifer (1994): Fuhr, N.; Pfeifer, U.: *Probabilistic information retrieval as a combination of abstraction, inductive learning, and probabilistic assumptions*. ACM Transactions on Information Systems (TOIS) **12**(1994)1, S. 92-115.

Gray/Reuter (1993): Gray, J.; Reuter, A.: *Transaction Processing. Concepts and Techniques*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

Greiff (1998): Greiff, W. R.: *A Theory of Term Weighting Based on Exploratory Data Analysis*. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, S. 11-19.

Han/Han/Choi (1993): Han, Y. S.; Han, Y. K.; Choi, K.: *Lexical concept acquisition from collocation map*. In: Workshop on Acquisition of Lexical Knowledge from Text, 31st Annual Meeting of the ACL, 1993.

Harper/Koychev/Sun/Pirie (2004): Harper, D. J.; Koychev, I.; Sun, Y.; Pirie, I.: *Within-Document Retrieval: A User-Centred Evaluation of Relevance Profiling*. Information Retrieval **7**(2004)3-4, S. 265-290.

Hermann/Hermann-Hasenmüller (1976): Hermann, W.; Hermann-Hasenmüller, U.: *Das Textinformationssystem STAIRS in der Praxis. Teil 1: Funktionen und Möglichkeiten.* IBM-Nachrichten **26**(1976), S. 236-242.

Hiemstra (2002): Hiemstra, D.: *Term-Specific Smoothing for the Language Modeling Approach to Information Retrieval: The Importance of a Query Term*. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2002, S. 35-41.

Hofmann (2001): Hofmann, T.: *Unsupervised learning by probabilistic latent semantic analysis*. Machine Learning **42**(2001)1-2, S. 177-196.

Hölscher (2002): Hölscher, C.: *Die Rolle des Wissens im Internet. Gezielt suchen und kompetent auswählen.* Stuttgart: Klett-Cotta, 2002.

Huffmann (1996): Huffmann, S. B.: *Learning information extraction patterns from examples*. Lecture Notes in Computer Science, Band 1040, 1996.

Hull (1996): Hull, D. A.: *Stemming algorithms: a case study for detailed evaluation*. Journal of the American Society for Information Science **47**(1996)1, S. 70-84.

Iwayama (2000): Iwayama, M.: *Relevance feedback with a small number of relevance judgements: incremental relevance feedback vs. document clustering*. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2000, S. 10-16.

Jansen (2005): Jansen, B. J.: *Seeking and implementing automated assistance during the search process*. Information Processing and Management **41**(2005)4, S. 909-928.

Jarosch (2003): Jarosch, H.: *Grundkurs Datenbankentwurf*. Wiesbaden: Friedr. Vieweg & Sohn, 2003.

Jarosch/Klein/Wulkau (1991): Jarosch, H.; Klein, R.; Wulkau, J.: *XTOOLS – an Artificial Intelligence Tool System*. In: Geske, U.; Koch, D. (Hrsg): Contributions to Artificial Intelligence. Berlin: Akademie-Verlag, 1991.

Jarosch/Müller (1979): Jarosch, H.; Müller, H.-D.: *Die Eigenschaften von Massendaten in automatisierten Dokumentennachweissystemen*. Dissertation. Ilmenau: THI, Fakultät für Technische Wissenschaften, 1979.

Jarosch/Müller (1984): Jarosch, H.; Müller, H.-D.: *Präzisierung von Rechercheanfragen im Dialog mit einem bibliographischen Datenbanksystem*. Dokumentation/Information (1984)62, S. 29-47.

Jarosch/Müller (1986): Jarosch, H.; Müller, H.-D.: *Programmierung interaktiver Informationsverarbeitungsprozesse in PL/I.* Berichte zur Wissenschaftsinformation und -kommunikation **9**(1986)4.

Jarosch/Müller (1990): Jarosch, H.; Müller, H.-D.: *Towards Optimal Interaction of Maschine-made and Human Specialists in Cooperative Task Performance*. Journal of New Generation Computer Systems **3**(1990)4, S. 353-370.

Järvelin/Kekäläinen/Niemi (2001): Järvelin, K.; Kekäläinen, J.; Niemi, T.: *Expansion Tool: Concept-Based Query Expansion and Construction*. Information Retrieval **4**(2001)3-4, S. 231-255.

Kaiser (1993): Kaiser, A.: *Intelligente Information Retrieval Systeme*. Nachrichten für Dokumentation **45**(1993), S. 157-162.

Kaiser (1997): Kaiser, A.: *A note on Intelligent Information Retrieval Tools in the World Wide Web*. In: Proceedings of the 20th Annual Conference of the „Gesellschaft für Klassifikation", 1997, S. 335-342.

Kämmerer (1974): Kämmerer, W.: *Einführung in mathematische Methoden der Kybernetik*. Berlin: Akademie-Verlag, 1974.

Kantor (2001): Kantor, P.: *Foundations of Statistical Natural Language Processing.* Information Retrieval **4**(2001)1, S. 80-81.

Karzauninkat (2003): Karzauninkat, S.: *Die Suchmaschinenlandschaft 2003: Wirtschaftliche und technische Entwicklungen*. In: Machill, M.; Welp, C. (Hrsg.): Wegweiser im Netz, 2003, S. 509-538.

Kekäläinen/Järvelin (2000): Kekäläinen, J.; Järvelin, K.: *The Co-Effects of Query Structure and Expansion on Retrieval Performance in Probabilistic Text Retrieval*. Information Retrieval **1**(2000)4, S. 329-344.

Kekäläinen/Järvelin (2003): Kekäläinen, J.; Järvelin, K.: *User-oriented evaluation methods for information retrieval: a case study based on conceptual models for query expansion*. In: Exploring artificial intelligence in the new millennium. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2003.

Kelly/Dollu/Fu (2005): Kelly, D.; Dollu, V. D.; Fu, X.: *The loquacious user: A document-independent source of terms for query expansion*. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2005, S. 457-464.

Khan/Khor (2004): Khan, S. M.; Khor, S.: *Enhanced Web Document Retrieval Using Automatic Query Expansion*. Journal of the American Society for Information Science and Technology **55**(2004)1, S. 29-40.

Kirkerud (1989): Kirkerud, B.: *Object-Oriented Programming with SIMULA*. International Computer Science Series. Boston, MA: Addison-Wesley, 1989.

Klix (1971): Klix, F.: *Information und Verhalten*. Berlin: Deutscher Verlag der Wissenschaften, 1971.

Klusch/Bergamaschi/Edwards/Petta (2003): Klusch, M.; Bergamaschi, S.; Edwards, P.; Petta, P. (Hrsg.): *Intelligent Information Agents: The Agentlink Perspective*. Berlin, ...: Springer Verlag, 2003.

Knuth (1972): Knuth, D. E.: *The Art of Computer Programming. Vol 1. Fundamental Algorithms*. Reading, ...: Addison-Wesley, 1972.

Krause (1992): Krause, J.: *Intelligentes Information Retrieval*. In: Kuhlen, R. (Hrsg.): Experimentelles und praktisches Information Retrieval. Konstanz: Universitätsverlag Konstanz, 1992.

Lafferty/Zhai (2001): Lafferty, J.; Zhai, C.: *Probabilistic relevance models based on document and query generation*. In: Workshop on Language Modeling and Information Retrieval, 2001.

Lam/Lai (2001): Lam, W.; Lai, K.-Y.: *A meta-learning approach for text categorization*. In: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2001, S. 303-309.

Lam-Adesina/Jones (2001): Lam-Adesina, A. M.; Jones, G. J. F.: *Applying summarization techniques for term selection in relevance feedback*. In: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2001, S. 1-9.

Lewandowski (2005): Lewandowski, D.: *Web Information Retrieval. Technologien zur Informationssuche im Internet*. Frankfurt a. M.: Deutsche Gesellschaft für Informationswissenschaft und Informationspraxis e. V., DGI-Schrift (Informationswissenschaft 7), 2005.

Lewis (1991): Lewis, D. D.: *Learning in intelligent information retrieval*. In: Proceedings of the 8th International Workshop on Machine Learning, 1991, S. 235-239.

Lewis/Catlett (1994): Lewis, D. D.; Catlett, J.: *Heterogeneous uncertainty sampling for supervised learning*. In: International Conference on Machine Learning (ICML), 1994.

Lewis/Ringuette (1994): Lewis, D. D.; Ringuette, M.: *A Comparison of Two Learning Algorithms for Text Classification*. In: Third Annual Symposium on Document Analysis and Information Retrieval, 1994, S. 81-93.

Lewis/Schapire/Callan/Papka (1996): Lewis, D. D.; Schapire, R. E.; Callan, J. P.; Papka, R.: *Training algorithms for linear text classifiers*. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1996, S. 298-306.

Liddy (1998): Liddy, E. D.: *Enhanced text retrieval using natural language processing*. Bulletin of the American Society for Information Science (1998)April/May, S. 14-16.

Liu/Croft (2004): Liu, X.; Croft, W. B.: *Statistical Language Modeling For Information Retrieval*. Annual Review of Information Science and Technology **39**(2004)1, S. 3-31.

Loney/Theriault (2001): Loney, K.; Theriault, M.: *Oracle9i DBA Handbook*. Berkeley, CA: Osborne/McGraw-Hill, 2001.

Lovins (1968): Lovins, J.: *Development of a Stemming Algorithm*. In: Mechanical Translation and Computational Linguistics **11**(1968)1-2, S. 22-31.

Magennis/van Rijsbergen (1997): Magennis, M.; van Rijsbergen, C. J.: *The potential and actual effectiveness of interactive query expansion*. In: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1997, S. 324-332.

Mano/Ogawa (2001): Mano, H.; Ogawa, Y.: *Selecting expansion terms in automatic query expansion*. In: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2001, S. 390-391.

Martelli/Montanari (1978): Martelli, A.; Montanari, U.: *Optimizing decision trees through heuristic guided search*. Communications of the ACM **21**(1978)12, S. 1025-1039.

Mayer (1978): Mayer, O.: *Syntaxanalyse*. Mannheim, Wien, Zürich: Bibliographisches Institut, 1978.

McCallum/Nigam (1998): McCallum, A. K.; Nigam, K.: *Employing EM and Pool-Based Active Learning for Text Classification*. In: Proceedings of the 15th International Conference on Machine Learning, 1998, S. 350-358.

McCallum/Nigam/Rennie/Seymore (1999): McCallum, A. K.; Nigam, K.; Rennie, J.; Seymore, K.: *A machine learning approach to building domain-specific search engines*. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999.

Michalski (1983): Michalski, R. S.: *A theory and methodology of inductive learning*. In: Michalski, R. S.; Carbonell, J. G.; Mitchell, T. M. (Hrsg): Machine Learning. An Artificial Intelligence Approach. Palo Alto, CA: Tioga Publishing Company, 1983, S. 83-134.

Michie/Spiegelhalter/Taylor (1994): Michie, D.; Spiegelhalter, D.; Taylor, C.: *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

Mitchell (1997): Mitchell, T. M.: *Machine Learning*. McGraw-Hill, 1997.

Mitchell/Caruana/Freitag/McDermott/Zabowski (1994): Mitchell, T.; Caruana, R.; Freitag, D.; McDermott, J.; Zabowski, D.: *Experience with a Learning Personal Assistent*. Communications of the ACM, **37**(1994)7, S. 81-91.

Mitra/Singhal/Buckley (1998): Mitra, M.; Singhal, A.; Buckley, C.: *Improving automatic query expansion*. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, S. 206-214.

Mittendorf/Mateev/Schäuble (2000): Mittendorf, E.; Mateev, B.; Schäuble, P.: *Using the Co-occurrence of Words for Retrieval Weighting*. Information Retrieval **3**(2000)3, S. 243-251.

Morgenroth (2006): Morgenroth, K.: *Kontextbasiertes Information Retrieval: Modelle, Konzeption und Realisierung kontextbasierter Information Retrieval Systeme*. Berlin: Logos-Verlag, 2006.

Muggleton (1996): Muggleton, S.: *Learning from positive data*. In: Muggleton, S. (Hrsg.): Proceedings of the 6th International Workshop on Inductive Logic Programming. Stockholm: Stockholm University, Royal Institute of Technologie, 1996, S. 225-244.

Müller/Thiel (1994): Müller, A.; Thiel, U.: *Query Expansion in an Abductive Information Retrieval System*. In: Proceedings of the RIAO '94, New York, S. 461-480.

Murdock/Croft (2005): Murdock, V.; Croft, W. B.: *A Translation Model for Sentence Retrieval*. In: Proceedings of the Conference on Human Language Technologies and Empirical Methods in Natural Language Processing (HLT/EMNLP), 2005, S. 684-691.

Nakashima/Sato/Qu/Ito (2003): Nakashima, M.; Sato, K.; Qu, Y.; Ito, T.: *Browsing-based conceptual information retrieval incorporating dictionary term relations, keyword association, and a user's interest*. Journal of the American Society for Information Science and Technology **54**(2003)1, S. 16-28.

Nallapati (2004): Nallapati, R.: *Discriminative models for information retrieval*. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004, S. 64-71.

Nallapati/Croft/Allan (2003): Nallapati, R.; Croft, W. B.; Allan, J.: *Relevant Query Feedback in Statistical Language Modelling*. In: Proceedings on the 12th International Conference on Information and Knowledge Management (CIKM '03), 2003, S. 560-563.

Ogawa/Morita/Kobayashi (1991): Ogawa, Y.; Morita, T.; Kobayashi, K.: *A fuzzy document retrieval system using the keyword connection matrix and a learning method*. In: Fuzzy Sets and Systems **39**(1991), S. 163-179.

Ojala (2002): Ojala, M.: *Web Search Engines: Search Syntax und Features*. Online **26**(2002)5, S. 27-32.

Papka/Allan (1998): Papka, R.; Allan, J.: *Document Classification using Multiword Features*. In: Proceedings on the 7th International Conference on Information and Knowledge Management (CIKM '98), 1998.

Parsay/Chignell/Khoshafian/Wong (1989): Parsay, K.; Chignell, M.; Khoshafian, S.; Wong, H.: *Intelligent Databases - Object-Oriented, Deductive, Hypermedia Technologies*. New York, NY: John Wiley & Sons, Inc., 1989.

Poetzsch (2005): Poetzsch, E.: *Information Retrieval. Einführung in Grundlagen und Methoden*. 4. Auflage. Potsdam: Verlag für Berlin-Brandenburg, 2005.

Ponte/Croft (1998): Ponte, J. M.; Croft, W. B.: *A Language Modeling Approach to Information Retrieval*. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998, S. 275-281.

Porter (1980): Porter, M. F.: *An Algorithm for Suffix Stripping*. Program **14**(1980)3, S. 130-137.

Quinlan (1986): Quinlan, J. R.: *Induction of Decision Trees*. Machine Learning Journal **1**(1986), S. 81-106.

Quinlan (1993): Quinlan, J. R.: *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1993.

Robertson (1990): Robertson, S. E.: *On Term Selection for Query Expansion*. Journal of Documentation **46**(1990)4, S. 359-364.

Russell/Norvig (2004): Russell, S.; Norvig, P.: *Künstliche Intelligenz. Ein moderner Ansatz*. 2. Auflage. München: Pearson Education, 2004.

Ruthven (2003): Ruthven, I.: *Re-examining the potential effectiveness of interactive query expansion*. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003, S. 213-220.

Ruthven/Lalmas/van Rijsbergen (2003): Ruthven, I.; Lalmas, M.; van Rijsbergen, K.: *Incorporating User Search Behavior into Relevance Feedback*. Journal of the American Society for Information Science and Technology **54**(2003)6, S. 529-549.

Sachs (1972): Sachs, L.: *Statistische Auswertungsmethoden*. Berlin, Heidelberg, New York: Springer-Verlag, 1972.

Salton/Buckley (1990): Salton, G.; Buckley, C.: *Improving retrieval performance by relevance feedback*. Journal of the American Society for Information Science **41**(1990)4, S. 288-297.

Schnauder/Jarosch/Mages (2004): Schnauder, V.; Jarosch, H.; Mages, M.: *Datenbankgestützte Vertriebs- und Informationssysteme*. Berlin: Logos Verlag, 2004.

Schnauder/Jarosch/Thieme (2001). Schnauder, V.; Jarosch, H.; Thieme, I.: *Praxis der Software-Entwicklung*. Renningen-Malmsheim: Expert Verlag, 2001.

SearchManager (2006): *SearchManager/370. Knowledge Management with the SearchManager/370*. IBM, 2006: http://www-306.ibm.com/software/data/sm370/about.html.

Sebesta (1996): Sebesta, R.: *Concepts of Programming Languages*. Boston, MA: Addison-Wesley, 1996.

Shah/Croft (2004): Shah, C.; Croft, W. B.: *Evaluating High Accuracy Retrieval Techniques*. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004, S. 2-9.

Shen/Zhai (2005): Shen, X; Zhai, C.: *Active feedback in ad hoc information retrieval*. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2005, S. 59-66.

Shute/Smith (1993): Shute, S. J.; Smith, P. J.: *Knowledge-based search tactics*. Information Processing and Management **29**(1993)1, S. 29-45.

Sommerville (2001): Sommerville, I.: *Software Engineering*. 6. Auflage. München: Pearson Studium, 2001.

Sormunen (2001): Sormunen, E.: *Extensions to the STAIRS Study – Empirical Evidence for the Hypothesised Ineffectiveness of Boolean Queries in Large Full-Text Databases*. Information Retrieval **4**(2001)3-4, S. 257-273.

Sparck Jones (1991): Sparck Jones, K.: *The Role of Artificial Intelligence in Information Retrieval*. Journal of the American Society for Information Science **42**(1991)8, S. 558-565.

Spink/Jansen/Ozmultu (2000): Spink, A.; Jansen, B. J.; Ozmultu, H. C.: *Use of query reformulation and relevance feedback by Excite users.* Internet Research: Electronic Networking Applications and Policy **10**(2000)4, S. 317-328.

Spink/Saracevic (1992): Spink, A.; Saracevic, T.: *Sources and Use of Search Terms in Online Searching*. In: Proceedings of the ASIS Annual Meeting of the American Society for Information Science **29**(1992), S. 249-255.

Spink/Saracevic (1997): Spink, A.; Saracevic, T.: *Interaction in information retrieval: selection and effectiveness of search terms*. Journal of the American Society for Information Science **48**(1997)8, S. 741-761.

Stahlknecht/Hasenkamp (2005): Stahlknecht, P.; Hasenkamp, U.: *Einführung in die Wirtschaftsinformatik*. 11. Auflage. Berlin, Heidelberg, New York: Springer-Verlag, 2005.

STAIRS (1981): *Storage and Information Retrieval System / Virtual Storage (STAIRS/VS). Operations Guide*. IBM: Order No. SH12-5500-6, 1981.

Stock (2000): Stock, W. G.: *Informationswirtschaft: Management externen Wissens*. München, ...: Oldenbourg, 2000.

Störrle (2005): Störrle, H.: UML 2 für Studenten. München u.a.: Pearson Education, 2005.

Strohman/Turtle/Croft (2005): Strohman, T.; Turtle, H.; Croft, W. B.: *Optimization strategies for complex queries*. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2005, S. 219-225.

Sturm (2002): Sturm, E.: *Das neue PL/I für PC, Workstation und Mainframe*. 5. Auflage. Wiesbaden: Friedr. Vieweg & Sohn, 2002.

Tong/Koller (2001) Tong, S.; Koller, D.: *Active learning: theory and applications*, 2001.

Turney (2000): Turney, P. D.: *Learning Algorithms for Keyphrase Extraction*. Information Retrieval **2**(2000)4, S. 303-336.

Unger/Wysotzki (1981): Unger, S.; Wysotzki, F.: *Lernfähige Klassifizierungssysteme*. Berlin: Akademie-Verlag, 1981.

Utgoff (1989): Utgoff, P. E.: *Incremental induction of decision trees*. Machine Learning Journal **4**(1989), S. 161-186.

Utgoff/Berkman/Clouse (1997): Utgoff, P. E.; Berkman, N. C.; Clouse, J. A.: *Decision tree induction based on efficient tree restructuring*. Machine Learning **29**(1997), S. 5-44.

van der Pol (2003): van der Pol, R.: *Dipe-D: A Tool for Konwledge-Based Query Formulation in Information Retrieval*. Information Retrieval **6**(2003)1, S. 21-47.

Vapnik (1999): Vapnik, V. N.: *The Nature of Statictical Learning Theory – Statistics for Engineering and Information Science*. 2. Auflage. Berlin, ...: Springer Verlag, 1999.

Vechtomova/Robertson/Jones (2003): Vechtomova, O.; Robertson, S.; Jones, S.: *Query Expansion with Long-Span Collocates*. Information Retrieval **6**(2003)2, S. 251-273.

Wen/Lao/Ma (2004): Wen, J.-R.; Lao, N.; Ma, W.-Y.: *Probabilistic model for contextual retrieval*. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004, S. 57-63.

Witten/Nevill-Manning/Maulsby (1996): Witten, I.; Nevill-Manning, C.; Maulsby, D.: *Interacting with Learning Agents: Implications for ML from HCI*. In: International Conference on Machine Learning (ICML), Workshop on Machine Learning meets Human Computer Interaction, 1996, S. 51-58.

Zhang/Chen (2002): Zhang, C.; Chen, T.: *An active learning framework for content-based information retrieval*. IEEE Transactions on Multimedia **4**(2002), S. 260-268.

Zoller (1981): Zoller, P.: *Abstrakte Datentypen*. Angewandte Informatik **23**(1981)10, S. 429-431.

Glossar

abstrakte Maschine: informationsverarbeitende Einheit, die ein individuelles Datenobjekt besitzt und dieses nach einem vorgegebenen Algorithmus bearbeitet.

Aktivator: abstrakte Maschine, die eine andere abstrakte Maschine zur Fortsetzung ihrer Arbeit auffordert.

Aktivator-Rückkehrpunkt: Sendepunkt, in dem eine abstrakte Maschine die Steuerung an ihren Aktivator zurückgibt.

Aktivierung: Vorgang, bei dem eine abstrakte Maschine zur Fortsetzung ihrer Arbeit aufgefordert wird.

Aktivierungspunkt: Sendepunkt, in dem eine abstrakte Maschine zur Fortsetzung ihrer Arbeit aufgefordert wird.

Austrittspunkt: letzter Schritt im Informationsverarbeitungsprozess eines KI-Assistenten, in dem der Benutzer die letzte Nachricht des KI-Assistenten erhält.

Begriffsdeklaration: Element der Anfrage zur Beschreibung eines Begriffs.

Begriffskonstrukt: Ensemble von Wortformen und Kollokationen zur Prüfung, ob in einem Dokument von einem bestimmten Begriff die Rede ist.

bekräftigte Endungsfolge: Suffix einer Wortform, das im nichtüberwachten Lernprozess als Endungsfolge bestätigt wurde.

Dialogpunkt: Schritt im Informationsverarbeitungsprozess eines KI-Assistenten, in dem der KI-Assistent mit dem Benutzer in Kommunikation tritt.

Dokumentenaussage: Aussage, von der im Bereich des Gesamtdokuments die Rede ist.

Eintrittspunkt: Anfangspunkt des Algorithmus der Monitormaschine, in dem der KI-Assistent die erste Nachricht des Benutzers entgegennimmt.

Endpunkt: Sendepunkt eines Algorithmus; erreicht eine abstrakte Maschine diesen Sendepunkt, so gibt sie ihr Datenobjekt auf und übergibt die Steuerung gemeinsam mit einer Nachricht an eine andere abstrakte Maschine.

Endungsfolge: Symbolfolge, die vollständig aus Elementen der Endungsmenge besteht.

Endungsmenge: Menge von Symbolfolgen, die jeweils elementare Endung einer Wortform sein können.

erweitertes Steuerprogramm: Steuerprogramm, das die Koordination des Zusammenspiels des Information-Retrieval-Systems, des Transaktions-Managers und der abstrakten Maschinen des KI-Assistenten übernimmt und Werkzeuge für moderne Programmiertechniken bereitstellt.

Feinanfrage: Suchanfrage an das Information-Retrieval-System als zweite Stufe des Klassifikators zum Thema des Benutzers.

Generator: abstrakte Maschine, die eine andere abstrakte Maschine ins Leben gerufen hat.

Generierung: Vorgang, bei dem eine neue abstrakte Maschine in den KI-Assistenten aufgenommen wird, wobei ihr zugleich die Steuerung übergeben wird.

Generierungspunkt: Sendepunkt, in dem eine neue abstrakte Maschine eingerichtet und aktiviert wird.

Grobanfrage: Suchanfrage an das Information-Retrieval-System als erste Stufe des Klassifikators zum Thema des Benutzers.

KI-Assistent: zeitabhängiges System abstrakter Maschinen, die nach den Algorithmen von Anwenderprogrammen arbeiten. Der KI-Assistent vermittelt zwischen dem Benutzer und dem Information-Retrieval-System und hebt dadurch die Mensch-Maschine-Kommunikation auf ein höheres Niveau.

Lernmenge: Menge von Fundorten aus relevanzbewerteten Dokumenten.

Lesekopf: Datentyp von KOMPROMISS zur Fixierung des erreichten Standes der lexikalischen Analyse eines Textes.

Monitormaschine: abstrakte Maschine, die die Regie über den Informationsverarbeitungsprozess eines KI-Assistenten übernimmt.

normierte Wortform: Symbolfolge, die aus einer Wortform durch Streichen einer bekräftigten Endungsfolge entsteht.

Objektmenge: Menge von Fundorten; für alle in der Objektmenge referierten Dokumente ist eine Entscheidung für die Klassen „ausgeben“ bzw. „nicht ausgeben“ zu fällen.

Parameter-Schnittstelle: ständig existierende globale Schnittstelle für den Austausch von Parametern zwischen den abstrakten Maschinen.

Präfixbaum: rekursive Datenstruktur zur Speicherung einer Wortmenge.

programmierter Dialog: Kommunikation zwischen den abstrakten Maschinen des KI-Assistenten und dem Information-Retrieval-System.

Prozess: Algorithmus, der vor dem Erreichen seines Endpunkts wenigstens einmal die Steuerung an den Aktivator zurückgibt.

Rest-Lernmenge: Teilmenge der Lernmenge, die nach Anwendung einer Selektionsoperation übrigbleibt.

Rest-Objektmenge: Teilmenge der Objektmenge, die nach Anwendung einer Selektionsoperation übrigbleibt.

Routine: Algorithmus, der nach dem Prinzip eines Unterprogramms arbeitet und erst in seinem Endpunkt die Steuerung an den Aktivator zurückgibt.

Satzaussage: Aussage, von der innerhalb eines Dokumenten-Satzes die Rede ist.

Schnittstelle: Speicherbereich für den Datenaustausch zwischen den abstrakten Maschinen des KI-Assistenten.

Segmentaussage: Aussage, von der im Bereich eines Dokumenten-Segments die Rede ist.

selektierte Lernmenge: Teilmenge der Lernmenge, die durch eine Selektionsoperation gewonnen wird.

selektierte Objektmenge: Teilmenge der Objektmenge, die durch eine Selektionsoperation gewonnen wird.

Sendepunkt: Schritt eines Algorithmus; führt eine abstrakte Maschine diesen Schritt aus, so übergibt sie die Steuerung gemeinsam mit einer Nachricht an eine andere abstrakte Maschine.

Teilsystem abstrakter Maschinen: Ensemble von abstrakten Maschinen, die gemeinsam zur Lösung einer abgegrenzten Aufgabe eingerichtet werden.

Teilsystem-Schnittstelle: zeitweilig existierende Schnittstelle für den Datenaustausch zwischen den abstrakten Maschinen eines Teilsystems.

Teilsystem von Komaschinen: Teilsystem abstrakter Maschinen, die hinsichtlich ihres Steuerungsverhaltens alle gleichberechtigt sind.

Unterprogramm-Aufrufpunkt: Sendepunkt, in dem eine abstrakte Maschine eingerichtet und aktiviert wird, die erst in ihrem Endpunkt die Steuerung an ihren Aktivator zurückgibt.

Wortform: konkrete Erscheinungsform eines Wortes im Text eines Dokuments.

zusätzliche Intelligenz: Anwenderprogramme, durch die die Mensch-Maschine-Kommunikation bei der Nutzung des Information-Retrieval-Systems auf ein höheres Niveau gehoben wird.