Inkrementelle, rückgekoppelte Suche in Software-Bibliotheken

Christian Lindig
Arbeitsgruppe Softwaretechnologie
Institut für Programmiersprachen
Technische Universität Braunschweig
Gaußstraße 17, D-38106 Braunschweig
lindig@ips.cs.tu-bs.de

November 1994

Abstract

Wiederverwendung von Software-Dokumenten verlangt eine Ablage, in der relevante Dokumente leicht aufgefunden werden können. Dokumente werden zur Suche in einer Sammlung durch Schlüsselwörter unabhängig von einander indexiert. Dokumente und Schlüsselwörter bilden zusammen einen formalen Kontext, der einen Begriffsverband impliziert. Der Begriffsverband stellt eine natürliche, nur aus der Indexierung abgeleiteten Gruppierung von Dokumenten und Attributen dar und erlaubt die effiziente inkrementelle Suche mit kontextsensitiver Unterstützung für den Benutzer. Die verschiedenen Operationen werden formal und anhand eines Beispiels präsentiert. Für eine Implementierung wird eine günstige Repräsentation des Begriffsverbandes vorgeschlagen und die typische Komplexität durch Experimente untersucht. Abschließend wird das präsentierte Verfahren mit verwandten Ansätzen verglichen.

1 Einleitung

Die Wiederverwendung von Software-Dokumenten wie Quelltexten, Designbeschreibungen, formalen und informalen Spezifikationen erfordert eine geordnete Ablage, aus der ein Benutzer geeignete Dokumente selektieren kann. Dokumente müssen leicht zu erfassen sein und die Suche muß jeweils die geeignetsten Dokumente selektieren, da das genau benötige Dokument oft nicht Teil der Sammlung ist [7]. Das hier präsentierte Verfahren zur Indexierung und Suche von Komponenten ist aus Sicht des Anwenders durch folgende Eigenschaften charakterisiert:

- Komponenten werden durch Schlüsselwörter manuell indexiert. Die Schlüsselwörter beschreiben die Komponente aus Sicht des Verfahrens vollständig – der eigentliche Inhalt bleibt darüberhinaus unberücksichtigt. Dadurch bleibt das Verfahren allgemein und kann auf verschiedene, auch inhomogene, Komponenten-Sammlungen angewendet werden.
- Komponenten einer Sammlung sind unabhängig von einander. Beim Einfügen und Löschen von Komponenten muß keine Beziehungen zwischen Komponenten spezifiziert oder berücksichtigt werden.

- Die Suche von Komponenten geschieht durch Angabe einer Menge von Schlüsselwörtern. Eine Menge von Schlüsselwörtern bestimmt eindeutig eine Menge von Komponenten.
- Die Suche von Komponenten ist effizient und erlaubt deshalb interaktives Arbeiten mit kurzen Antwortzeiten.
- Die Suche ist inkrementell. Eine Anfrage kann schrittweise spezialisiert werden und zuvor berechnete Ergebnisse können für die Bearbeitung der neuen Anfrage weiterverwendet werden.
- Das Verfahren kann zu jeder Anfrage effizient und vollständig bestimmen, wie die Anfrage weiter spezialisiert werden kann. Dem Benutzer können also alle *sinnvollen* Vorschläge zur Verschärfung seiner Anfrage präsentiert werden – der Benutzer wird kontextsensitiv unterstützt.
- Der Benutzer ist nicht auf die Suche einzelner Komponenten beschränkt. Er kann einen stufenlosen Übergang zwischen der Suche einzelner Komponenten und dem Selektieren einer Teilmenge von Komponenten wählen.

Die Eigenschaften des Suchverfahrens basieren auf einer Analyse der Beziehungen zwischen Komponenten durch formale Begriffsanalyse. Formale Begriffsanalyse [19] gruppiert die Komponenten in einer aus ihren Attributen abgeleiteten natürlichen Weise zu Begriffen, die einen Verband bilden.

Der nächste Abschnitt gibt zunächst einen Überblick über das Verfahren in Form eines Beispiels. Die sich daran anschließenden Abschnitte definieren Begriffsanalyse formal und zeigen, wie sie zur Suche eingesetzt werden kann. Implementierungsaspekte und ein Vergleich mit anderen Ansätzen bilden den Schluß.

2 Beispiel Unix-System-Calls

Zur Entwicklungsumgebung unter Unix gehört die C-Bibliothek `libc`. Sie enthält viele wiederverwendbare Komponenten, aber das Auffinden dieser Komponenten wird von der Entwicklungumgebung unzureichend unterstützt. Die Bibliothek enthält Funktionen zur String-Behandlung, Ein-/Ausgabe, Prozeßkommunikation und auch die Funktionen zum Aufruf von Betriebssystem-Funktionen (System-Calls). Die Dokumentation der Funktionen dieser und anderer Bibliotheken und der System-Calls umfaßt etwa 1500 Dokumente und kann mit Hilfe des `man`-Kommandos (für Manual) selektiert werden. Zur Suche von Funktionen, bzw. ihrer Dokumentation steht nur eine Teilstringsuche in einer Liste von einzeiligen Beschreibungen der jeweiligen Dokumentation zur Verfügung.

Tabelle 1 zeigt die einzeilige Beschreibung von sieben System-Calls (von insgesamt ca. 180). Sie sollen neu mit Schlüsselwörtern indexiert werden und als prinzipielles Beispiel für eine spätere Suche dienen. Jedem System-Call werden eine Menge von charakteristischen Schlüsselwörtern zugeordnet – die neue Zuordnung zeigt Tabelle 2.

Jeder Komponente können eine beliebige Anzahl von Schlüsselwörtern zugeordnet werden. Die Schlüsselwörter sollen eine klare Semantik besitzen, die der Anwender der Suche kennen muß. Im Falle der Unix-System-Calls existieren für viele Objekte eindeutige Namen,

`audit()`	*write a record to the audit log*
`auditon()`	*manipulate auditing*
`auditsvc()`	*write audit records to specified file descriptor*
`bind()`	*bind a name to a socket*
`brk()`	*change data segment size*
`chdir()`	*change current working directory*
`chmod()`	*change mode of file*

Tabelle 1: Unix-System-Calls und ihre Kurzbeschreibung

`audit()`	$\{write, record, log\}$
`auditon()`	$\{enable, control, audit, log\}$
`auditsvc()`	$\{write, audit, file\}$
`bind()`	$\{bind, associate, name, socket\}$
`brk()`	$\{change, segment, process\}$
`chdir()`	$\{change, directory, filesystem\}$
`chmod()`	$\{change, permissions, file, handle\}$

Tabelle 2: Indexierung mit Schlüsselwörtern

so daß die Indexierung oft naheliegend ist: Zeichenketten heißen immer *strings* und Zugriffsrechte für Dateien *permissions*. Im Allgemeinen ist dies so nicht der Fall und die Indexierung von Komponenten ist ein wesentlicher Punkt beim Aufbau einer Komponenten-Sammlung.

Zur Suche von Komponenten gibt der Anwender eine Menge von Schlüsselwörtern an – wegen der Mengencharakteristik ist die Reihenfolge der Wörter beliebig. Aus der Menge aller Komponenten werden die Komponenten ausgewählt, deren Schlüsselwortmenge die angegebenen Schlüsselwörter mindestens umfassen. Dies erlaubt es, aus der Menge aller Komponenten eine Untermenge beliebig durch ein ein oder mehrere Schlüsselwörter zu selektieren: Tabelle 3 zeigt Schlüsselwortmengen und die Mengen der jeweils selektierten Komponenten. Das Ergebnis einer Anfrage enthält genau die durch die Anfrage beschriebenen Komponenten und ist dadurch für den Anwender plausibel; die Anfrage wird nicht abgeschwächt oder umgeformt, um „verwandte“ Komponenten zu finden.

Schlüsselwortmenge	selektierte Komponenten
$\{\}$	alle Komponenten
$\{file\}$	$\{chmod(), auditsvc()\}$
$\{file, change\}$	$\{chmod()\}$
$\{change\}$	$\{chdir(), brk(), chmod()\}$
$\{log, name\}$	$\{\}$

Tabelle 3: Suche durch Mengen von Schlüsselwörtern

Die Komponente *chmod()* wird bereits durch die Angabe der beiden Attribute *file* und

change eindeutig selektiert, obwohl sie noch mit weiteren Schlüsselwörtern indexiert ist. Um eine Komponente eindeutig zu selektieren, muß also nicht zwingend ihre vollständige Schlüsselwortmenge angegeben werden. Die Selektivität einer Anfrage hängt von der Ausprägung der Sammlung ab.

Da die Suche effizient ist, kann der Anwender seine Anfrage häufig variieren und durch Erweitern der Schlüsselwortmenge spezialisieren und sich so von einem *beliebigen* Ausgangspunkt einer kleinen interessanten Auswahl von Komponenten nähern. Zu jeder Anfrage läßt sich effizient bestimmen, welche Attribute für eine Verschärfung der Anfrage überhaupt noch in Frage kommen. Der Anwender ist also nicht auf Raten angewiesen, sondern erhält eine starke kontextsensitive Hilfe bei der Reformulierung seiner Anfrage. Bei einer Anfrage sind noch genau die Schlüsselwörter zur Verschärfung möglich, die in den Schlüsselwort-Mengen der bereits selektierten Komponenten enthalten sind, aber noch nicht in der Anfrage selbst.

Die Anfrage $\{change\}$ selektiert $\{chdir(), brk(), chmod()\}$ und kann durch eines der Wörter aus $\{filesystem, directory, segment, process, handle, permissions\}$ weiter spezialisiert werden. Der Anwender kann nun zum Beispiel *directory* auswählen und erhält mit $\{change, directory\}$ die Komponente $\{chdir()\}$. Es stehen dem Anwender also nicht mehr alle möglichen Schlüssselwörter, sondern nur noch die sinnvollen zur Verfügung.

Die Grundlage dafür, daß der Benutzer die Sammlung der Komponenten von einem beliebigen Thema her schrittweise einschränken kann und ihm die jeweiligen Spezialisierungsmöglichkeiten auch bekannt sind, ist die geschickte interne Ablage der Komponenten. Formale Begriffsanalyse bestimmt aus den Komponenten und ihrer Indexierung einen Begriffsverband, der die dargestellte Navigation durch die Komponenten effizient erlaubt.

3 Formale Begriffsanalyse

Formale Begriffsanalyse wurde von R. Wille begründet und untersucht die Beziehung zwischen einer Menge von Objekten und diesen Objekten zugeordneten Attributen [19, 2].

Definition 1 *Ein formaler Kontext ist ein Tripel (O, A, R) aus einer endlichen Objektmenge O, einer endlichen Attributmenge A und einer Relation $R \subseteq O \times A$ zwischen ihnen. $(o, a) \in R$ wird gelesen als: Objekt o besitzt das Attribut a.*

Die Relation zwischen Objekten und Attributen läßt sich anschaulich in einer *Kontexttabelle* darstellen. Tabelle 4 zeigt die Kontexttabelle des Beispiels.

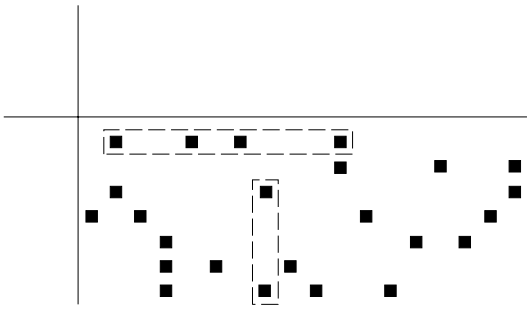

Tabelle 4: Kontext für Unix-System-Calls

Definition 2 *Für Objekt-Mengen $O' \subseteq O$ und Attributmengen $A' \subseteq A$ eines Kontextes (O, A, R) werden die Mengen der ihnen gemeinsamen Attribute bzw. Objekte durch Abbildungen ω und α definiert:*

$$\begin{array}{ll} \alpha : 2^A \rightarrow 2^O & \alpha(A') = \{o \in O | \forall a \in A' : (o, a) \in R\} \\ \omega : 2^O \rightarrow 2^A & \omega(O') = \{a \in A | \forall o \in O' : (o, a) \in R\} \end{array}$$

Die Objekte *chdir()* und *chmod()* besitzen nur *change* als gemeinsames Attribut, also $\omega(\{chdir, chmod\}) = \{change\}$.

Die zuvor definierten Funktionen α und ω bilden eine Galois-Verbindung zwischen den Mengen A und O, d.h. es gilt für $O', O'' \subseteq O; A', A'' \subseteq A$:

$$\begin{array}{l} A' \subseteq A'' \Rightarrow \alpha(A') \supseteq \alpha(A'') \\ O' \subseteq O'' \Rightarrow \omega(O') \supseteq \omega(O'') \\ A' \subseteq \omega(\alpha(A')) \\ O' \subseteq \alpha(\omega(O')) \end{array} \tag{1}$$

Definition 3 *Ein formaler Begriff eines Kontextes (O, A, R) ist ein Paar (O', A') aus Objekt- und Attributmengen mit $O' \subseteq O, A' \subseteq A$, wobei gleichzeitig gilt:*

$$\alpha(A') = O' \quad \textit{und} \quad \omega(O') = A'$$

O' heißt Umfang von (O', A'), A' Inhalt. Die Menge aller Begriffe eines Kontextes (O, A, R) wird mit $B(O, A, R)$ bezeichnet.

Die Objekte eines Begriffs (in einem gegebenen Kontext) sind synonym für eine Menge von Attributen und umgekehrt. Ein Beispiel für einen Begriff in dem Kontext Unix-System-Calls ist das folgende Paar:

$$(\{chmod(), auditsvc()\}, \{file\})$$

In dem gegebenen Kontext ist das Attribut *file* ein Synonym für die Funktionen *chmod()* und *auditsvc()* – und umgekehrt.

Definition 4 *Zwei Begriffe $(O_1, A_1), (O_2, A_2) \in B(O, A, R)$ eines Kontextes werden durch die folgende Relation $\leq$ geordnet:*

$$(O_1, A_1) \leq (O_2, A_2) :\Leftrightarrow O_1 \subseteq O_2$$

Die Relation wird gelesen als (O_1, A_1) ist Unterbegriff von (O_2, A_2). Das Paar aus Kontext und Ordnungsrelation wird abgekürzt als $\mathcal{B}(O, A, R) = (B(O, A, R), \leq)$.

In der Definition kann $O_1 \subseteq O_2$ auch durch $A_1 \supseteq A_2$ ersetzt werden, denn dies ist äquivalent wegen (1). Die Begriffe eines Kontextes werden durch eine (partielle) Ordnung Ober-/Unterbegriff geordnet. So ist in dem gegebenen Beispiel der Begriff $(\{chmod(), auditsvc()\}, \{file\})$ ein Oberbegriff des Begriffes $(\{auditsvc()\}, \{file, write, audit\})$. Ein Oberbegriff hat größeren Umfang und kleineren Inhalt als seine Unterbegriffe.

Theorem 1 (Hauptsatz der Begriffsanalyse [19]) *Sei $K = (O, A, R)$ ein Kontext, so ist $\mathcal{B}(O, A, R)$ ein vollständiger Verband, der* Begriffsverband *von K. Infimum und Supremum sind gegeben durch:*

$$\begin{aligned} \bigwedge_{i \in I}(O_i, A_i) &= (\bigcap_{i \in I} O_i \,,\, \omega(\alpha(\bigcup_{i \in I} A_i))) \\ \bigvee_{i \in I}(O_i, A_i) &= (\alpha(\omega(\bigcup_{i \in I} O_i)) \,,\, \bigcap_{i \in I} A_i) \end{aligned}$$

Dieser Satz besagt, daß zwei (oder mehr) Begriffe einen eindeutig bestimmten kleinsten Oberbegriff und größten Unterbegriff besitzen und beschreibt, wie diese Begriffe berechnet werden. Der größte gemeinsame Unterbegriff (Infimum) entsteht aus dem Schnitt der Objekt-Mengen und der Vereinigung der Attributmengen der beteiligten Begriffe, wobei die vereinigten Attributmengen noch erweitert werden müssen, damit sie mit dem Schnitt der Objektmengen wirklich einen Begriff bilden. Mit dem Supremum verhält es sich umgekehrt analog.

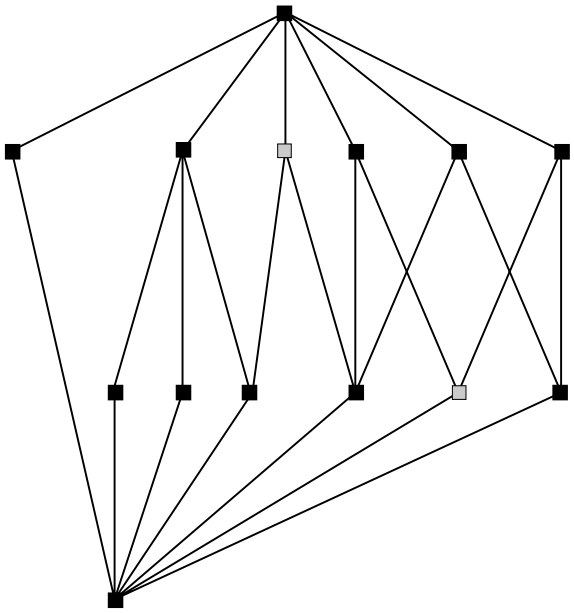

Abbildung 1: Hasse-Diagramm des Begriffsverbands für Unix-System-Calls

Abbildung 1 zeigt das Hasse-Diagramm des Begriffsverbandes des Beispiels. Jeder Knoten des Graphen ist ein Begriff, bestehend aus einem Paar aus einer Menge von System-Calls und einer Menge von Schlüsselwörtern. Statt dieser Paare sind in Abbildung 1 vereinfachte Beschriftungen an den Knoten angebracht, aus denen die jeweiligen Mengen aber rekonstruiert werden können:

- Attribute werden in Begriffsverbänden an Unterbegriffe vererbt. Alle kleineren Begriffe als der mit *file* beschriftete Begriff tragen ebenfalls *file* in ihrer Attributmenge.
- Objekte werden in Begriffsverbänden an Oberbegriffe vererbt. Alle Oberbegriffe des mit *auditsvc()* beschrifteten Begriffs tragen dieses Objekt ebenfalls in ihrer Objektmenge.

Definition 5 *Sei $\mathcal{B}(O, A, R)$ ein Begriffsverband und $o \in O$, $a \in A$. Seien folgende zwei Funktionen definiert:*

$$\sigma(o) : O \to B(O, A, R) \quad \sigma(o) = (\alpha(\omega(\{o\})), \omega(\{o\}))$$
$$\mu(a) : A \to B(O, A, R) \quad \mu(a) = (\alpha(\{a\}), \omega(\alpha(\{a\})))$$

Die Funktion σ gibt zu jedem Objekt o den kleinsten Begriff an, der o in seinem Umfang enthält. Die Funktion μ gibt zu jedem Attribut a den größten Begriff an, der a in seinem Inhalt enthält.

Der mit *file* beschriftete Begriff ist also das Paar $(\{chmod(), auditsvc()\}, \{file\})$. Zur Beschriftung der Knoten im Diagramm wurden die Funktionen μ und σ verwendet. Auf jedes Objekt und jedes Attribut wurde σ bzw. μ angewendet und das Objekt bzw. Attribut an dem sich ergebenden Begriff angetragen: $\mu(file)$ ist der mit *file* beschrifteten Knoten im Hasse-Diagramm.

Anschaulich betrachtet ist ein Begriff ein maximales Rechteck in der zugehörigen Kontexttabelle, wobei Spalten und Zeilen zur Bildung des Rechtecks vertauscht werden dürfen. In Abbildung 1 sind zwei Begriffe hervorgehoben, ebenso die zugehörigen maximalen Rechtecke in der Kontexttabelle 4.

Der größte gemeinsame Unterbegriff einer Menge von Begriffen ergibt sich in dem Diagramm durch das Verfolgen von Kanten nach unten zu einem gemeinsamen Begriff, analog dazu der kleinste gemeinsame Oberbegriff durch Verfolgen der Kanten nach oben.

Zur Berechnung aller Begriffe $B(O, A, R)$ eines Kontextes existieren mehrere Algorithmen [4]. Im schlechtesten Fall ist die Komplexität des Algorithmus exponentiell, da ein Kontext maximal 2^n mit $n = min(|O|, |A|)$ Begriffe enthalten kann. Auf den Aspekt der typischen Laufzeit wird noch weiter unten eigegangen.

4 Suche mit Begriffsverbänden

Die Komponenten O einer Sammlung bilden zusammen mit der Menge aller Schlüsselwörter A und der Relation R zwischen ihnen einen formalen Kontext (O, A, R) und damit einen Begriffsverband $\mathcal{B}(O, A, R)$.

Definition 6 *Zu einem Objekt $o \in O$ eines formalen Kontexts (O, A, R) beschreibt $\gamma(o)$ die Attribute von o:*

$$\gamma(o) := \{a \in A | (o, a) \in R\} \quad \gamma : O \to 2^A$$

Eine Anfrage ist eine Menge von Attributen $A' \subseteq A$; sie beschreibt alle Komponenten $o \in O$, deren Attribute Obermenge von A' sind: $\{o | o \in O, \gamma(o) \supseteq A'\}$. Dies beschreibt jedoch noch nicht, wie diese Menge bestimmt werden kann.

Jedes Objekt o und jedes Attribut a wird durch je einen Begriff $\sigma(o)$, bzw. $\mu(a)$ eingeführt und dann an größere bzw. kleinere Begriffe vererbt. Die in der Anfrage angegebenen Attribute A' bestimmt also eine Menge $\phi(A')$ von sie einführenden Begriffen in $B(O, A, R)$:

$$\phi(A') := \{\mu(a) | a \in A'\} \quad \phi : 2^A \to 2^{B(O,A,R)}$$

Der größte gemeinsame Unterbegriff $\chi(A')$ der Begriffe in $\phi(A')$ ist der größte (allgemeinste) Begriff, der alle Attribute aus A' trägt. Er ist das Infimum der Begriffe in $\phi(A')$:

$$\chi(A') = (O_\chi, A_\chi) = \bigwedge_{b \in \phi(A')} b \quad \chi : 2^A \to B(O, A, R)$$

Der Begriff $\chi(A') = (O_\chi, A_\chi)$ umfaßt alle Objekte, die mindestens die Attribute A' tragen. Damit ist O_χ die gesuchte Menge von Objekten zu der Anfrage A'. Da $\chi(A')$ der größte Begriff in $\mathcal{B}(O, A, R)$ mit dieser Eigenschaft ist, gibt es keinen größeren Begriff als $\chi(A')$, mit dieser Eigenschaft und insbesondere keinen Begriff mit einer größeren Anzahl von Objekten mit der geforderten Eigenschaft. Abbildung 2 zeigt noch einmal graphisch, wie der Begriff $\chi(A')$ ermittelt wird.

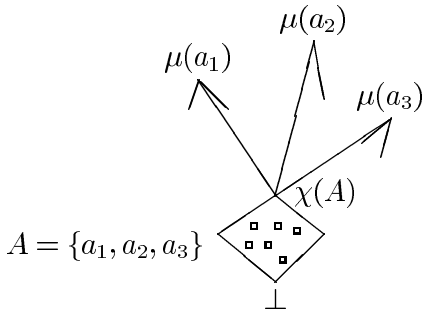

Abbildung 2: Suche im Begriffsverband

Jeder Begriff des Begriffsverbandes $\mathcal{B}(O, A, R)$ enthält alle ihn umfassenden Objekte und Attribute. In Abbildung 1 sind stattdessen an jeden Begriff nur durch diesen Begriff eingeführte Objekte und Attribute angetragen.

Definition 7 *Sei $b = (O_b, A_b) \in B(O, A, R)$ ein Begriff eines formalen Kontextes.*

$$\begin{aligned} \nu_a(b) &= \{a \in A \mid \mu(a) = b\} \\ \nu_o(b) &= \{o \in O \mid \sigma(o) = b\} \\ \pi_o(b) &= O_b \\ \pi_a(b) &= A_b \end{aligned}$$

Dann heißen $\nu_a(b)$ durch b neu eingeführten Attribute und $\nu_o(b)$ durch b neu eingeführte Objekte. Die Funktionen π_o und π_a sind Projektionen des Paares b auf seine Komponenten und liefern die Objektmenge bzw. Attributmenge eines Begriffes.

Die von der Anfrage A' beschriebene Menge von Objekten läßt sich auch allein durch die von Begriffen eingeführten Objekte ν_o beschreiben. Dies erlaubt bei einer Repräsentation von $\mathcal{B}(O, A, R)$ statt der vollständigen Begriffe kleinere Paare aus jeweils neu eingeführten Objekten und Attribute zu verwenden. Zur Berechnung von O_χ müssen die durch Unterbegriffe neu eingeführten Objekte vereinigt werden:

$$\pi_o(\chi(A')) = O_\chi = \bigcup_{b \leq \chi(A')} \nu_o(b) \tag{2}$$

Wenn $\chi(A')$ nicht der kleinste Begriff des Verbandes $\mathcal{B}(O, A, R)$ ist, sind die Unterbegriffe von $\chi(A')$ zusätzlich Unterbegriffe weiterer Begriffe, die mit $\chi(A')$ unvergleichlich sind. Dies bedeutet, daß sie Attribute tragen, die nicht in der Attributmenge $\pi_a(\chi(A'))$ auftreten. Die Attribute von Unterbegriffen, die Objekte einführen, und die nicht in der Attributmenge von $\chi(A')$ enthalten sind, verdienen besondere Beachtung: es sind genau die Attribute, die eine Anfrage A' sinnvoll spezialisieren können. Sinnvoll heißt hier, daß durch Hinzunahme eines Attributes in die Menge A' eine Untermenge der bisher selektierten Objekte ausgewählt wird – die Auswahl also spezieller wird. Die Menge der Attribute, die A' jeweils durch ihre Aufnahme sinnvoll spezialisieren, soll mit $\Theta(A')$ bezeichnet werden:

$$\Theta(A') := \Big(\bigcup_{b \leq \chi(A'), \nu_o(b) \neq \{\}} \pi_a(b) \Big) \setminus A' \tag{3}$$

Dies sind alle Attribute von Unterbegriffen von $\chi(A')$, die einen tatsächlichen Beitrag ($\nu_o(b) \neq \{\}$) zur Objektmenge von $\chi(A')$ leisten und die noch nicht in A' enthalten sind[1]. Die Menge $\Theta(A')$ ist also die Menge von Attributen, die einem Benutzer zur sinnvollen Verschärfung seiner Anfrage angeboten werden können.

Wählt der Benutzer zur Verschärfung seiner Anfrage eines der sinnvollen Attribute $a \in \Theta(A')$, so ist $A'' = A' \cup \{a\} \supset A'$ und es gilt $\chi(A'') \leq \chi(A')$. Zur Berechnung von $\chi(A'')$ genügt es, das Infimum von $\chi(A')$ und dem Begriff zu bilden, der das sinnvolle Attribut a einführt:

$$\chi(A'') = \mu(a) \wedge \chi(A')$$

Die Berechnung einer spezialisierten Anfrage kann also das Ergebnis der schwächeren Anfrage verwenden und ist somit inkrementell. Dies erlaubt für praktische Anwendungen eine schnellere Berechnung des Ergebnisses. Wählt der Benutzer statt eines Attributs eine Teilmenge $\theta \subseteq \Theta$, so ist nicht garantiert, daß die erweiterte Anfrage $A' \cup \theta$ eine nichtleere Menge von Objekten beschreibt, da sich die Attribute in Θ in einem speziellen Kontext ausschließen können.

5 Implementierungsaspekte

Für eine konkrete Realisierung des bisher beschriebenen Verfahrens gilt es die notwendigen Operationen möglichst effizient hinsichtlich Zeit- und Platzbedarf zu implementieren. Hierfür sind mehrere Ansätze denkbar, von denen hier einer als Kompromiß zwischen optimalem Speicher- und Zeitbedarf vorgestellt werden soll. Zur Berechnung der einzelnen Ergebnisse sind folgende primitive Operationen nötig:

- Zu einer Anfrage A' soll der größte Begriff $b \in B(O, A, R)$ bestimmt werden, dessen Attribute A' enthalten. Dazu müssen zunächst die Begriffe $\mu(a)$ bestimmt werden, die die einzelnen Attribute $a \in A'$ einführen und von diesen Begriffen muß dann das Infimum ($\wedge$) gebildet werden.

- Zur Bestimmung der gesuchten Objektmenge einer Anfrage muß zu dem gefundenen Begriff b seine Objektmenge $\pi_o(b)$ bestimmt werden.

[1] Tatsächlich würde es genügen, die echten Unterbegriffe von $\chi(A')$ zu betrachten. Da aber $\pi_a(b) \supseteq \pi_a(\chi(A'))$ für $b < \chi(A')$ gilt, ist das Ergebnis dasselbe.

- Zu jeder Anfrage soll die Menge der Objekte bestimmt werden, die die Anfrage sinnvoll erweitern. Dazu müssen für ein Objekt alle Unterobjekte ($\leq$) und von diesen alle Attributmengen (π_a) und neu eingeführte Objekte (ν_o) bestimmt werden.

5.1 Darstellung des Begriffsverbandes

Zwischen jedem Begriff b eines Begriffsverbandes und seiner Menge von Unterbegriffen $B_{\leq}(b)$ existiert eine Bijektion ρ die es erlaubt, das Infimum einer Menge von Begriffen zu bestimmen [1]:

$$\bigwedge_{b \in B} b = \rho^{-1}(\bigcap_{b \in B} B_{\leq}(b))$$

Der Schnitt der Unterbegriffs-Mengen der beteiligten Begriffe ergibt eine Menge von Begriffen, die genau einen Begriff des Begriffsverbandes beschreibt. Damit kann die Infimums-Bildung auf den Schnitt von Mengen zurückgeführt werden. Da zu jedem Begriff seine Unterbegriffe sowieso benötigt werden, bietet sich die folgende Darstellung des Begriffsverbandes an:

Begriff	$\leq$	$\nu_o(b)$	$\pi_a(b)$
b_0	$\{b_0^1, b_0^2, \ldots, b_0^m\}$	$\{o_0^1, o_0^2, \ldots, o_0^n\}$	$\{a_0^1, a_0^2, \ldots, a_0^l\}$
$\ldots$	$\ldots$	$\ldots$	$\ldots$

Zu jedem Begriff wird die reflexiv abgeschlossene Menge seiner Unterbegriffe ($\leq$), die von ihm eingeführten Objekte (ν_o) und seine vollständige Attributmenge (π_a) abgespeichert. Statt zu jedem Begriff seine vollständige Objektmenge $\pi_o(b)$ abzuspeichern, wird für jedes Objekt die deutlich kleinere Menge $\nu_o(b)$ abgespeichert. Um alle Objekte eines Begriffes zu erhalten, müssen dann die Unterbegriffe besucht werden und die von ihnen neu eingeführten Objektmengen vereinigt werden:

$$\pi_o(b) = \bigcup_{b' \leq b} b'$$

Um für jedes Attribut a den einführenden Begriff $\mu(a)$ zu erhalten, wird eine zweite Tabelle angelegt, die genau diese Information enthält und in die erste Tabelle verweist:

Attribut	$\mu(a)$
a_0	b_i
$\ldots$	$\ldots$

Statt zu jedem Begriff b die zugehörige Menge von Attributen $\pi_a(b)$ zu speichern, könnten auch nur die durch b eingeführten Attribute $\nu_a(b)$ abgelegt werden. Dann müßte aber zum Bestimmen von $\pi_a(b)$ alle größeren Begriffe als b besucht werden. Dazu müßte zu jedem Begriff auch die reflexive abgeschlossene Menge seiner Vorgänger gespeichert werden, die ansonsten nicht benötigt wird. Deshalb wird darauf verzichtet und zu jedem Begriff seine vollständige Menge von Attributen $\pi_a(b)$ abgelegt, wie es in der obige Tabelle auch angegeben ist.

Mit der gewählten Darstellung können alle erforderlichen Operationen leicht implementiert werden:

- Der zu einer Anfrage A' gehörende Begriff $\chi(A')$ wird über die spezifizierten Attribute $a \in A'$ bestimmt: zu jedem a wird über die zweite Tabelle ein Begriff in der ersten Tabelle bestimmt; dessen Nachfolgermengen werden geschnitten. Die sich ergebende Menge beschreibt genau den gesuchten Begriff in der ersten Tabelle.
- Zu jedem Begriff b (in der ersten Tabelle) können alle zugehörigen Objekte $\pi_o(b)$ bestimmt werden, indem alle seine Nachfolger besucht und die Mengen ν_o vereinigt werden.
- Zur Bestimmung der sinnvollen weiteren Attribute einer Anfrage nach (3) stehen die nötige Mengen der Unterbegriffe, ν_o und π_a ebenfalls unmittelbar zur Verfügung.

Die gewählte Repräsentation erlaubt also eine effiziente Berechnung der relevanten Operationen.

5.2 Berechnung von Begriffsverbänden

Das präsentierte Beispiel ist sehr klein und zeigt lediglich das Prinzip. Um die praktische Relevanz beurteilen zu können, sind aber Angaben über die typische Größe des entstehenden Begriffsverbandes und den Aufwand ihn zu bestimmen unerläßlich. Deshalb wurden einige Experimente mit zufällig generierten Kontexten unterschiedlicher Größe ausgeführt.

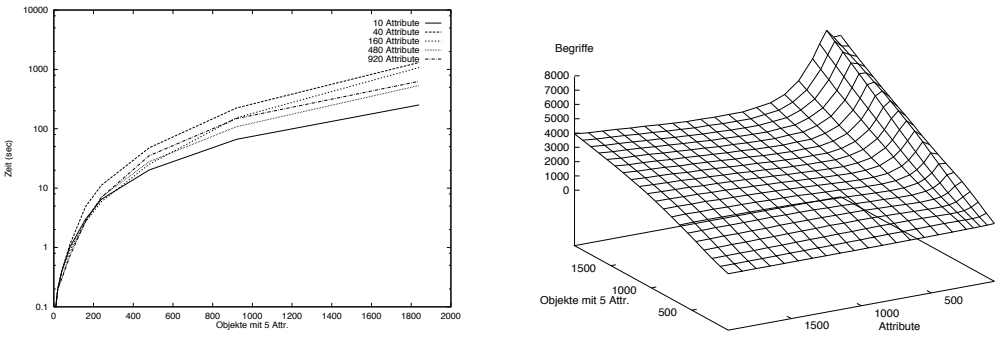

Abbildung 3: Zeitaufwand zur Berechnung von Begriffsverbänden und ihre Größe

Für einen zufällig generierten Kontext wurde jeweils die Zeitdauer zur Berechnung des zugehörigen Begriffsverbandes und die Anzahl der Begriffe in dem Verband bestimmt. Jeder Kontext enthält eine bestimmte Anzahl von Objekten, die mit genau fünf Attributen versehen sind. Die Entscheidung jedes Objekt mit fünf Attributen zu versehen ist zunächst willkürlich. Sie wurde gewählt, um eine facettierte Klassifikation mit fünf Facetten nachzubilden [14]. Die Attribute wurden zufällig aus einer bestimmten Anzahl von Attributen ausgewählt. Dies entspricht einer Kontexttabelle mit bestimmter Zeilen- und Spaltenanzahl, in deren Zeilen genau fünf Kreuze stehen. Die Anzahl der Objekte und die Größe der Attributmenge variierte zwischen den generierten Kontexten. Die verschiedenen Kontexte simulieren die Indexierung von verschiedenen Anzahlen von Objekten mit einem unterschiedlich großen Wortschatz, aus dem je fünf Begriffe ausgewählt wurden.

Zur Berechnung der Begriffsverbände wurde ein in C geschriebenes Programm eingesetzt, das den von Ganter entwickelten Algorithmus implementiert [4]. Das Programm lief auf einer Sun SPARCstation ELC; die gemessenen Zeiten in Sekunden geben die reale

Rechenzeit, gemessen mit dem Unix `time`-Kommando wieder. Abbildung 3 zeigt den ermittelten Zeitaufwand (mit logarthmischer Zeitachse) und die Größe der enstehenden Begriffsverbände.

Der Zeitaufwand bei den untersuchten Kontexten ist im Wesentlichen durch die Zahl der Objekte, also die Größe einer Sammlung, bestimmt. Die Größe des Wortschatzes zur Indexierung hat dagegen geringeren Einfluß. Die gemessenen Rechenzeiten liegen zwischen einigen Sekunden für mehrere hundert Objekte und steigen bis zu mehreren Minuten für Sammlungen von fast 2000 Objekten an. Die Größe der enstandenen Begriffsverbände wird ebenfalls durch die Zahl der Objekte dominiert, allerdings entstehen auffallend große Verbände bei vielen Objekten mit vergleichsweise wenigen Attributen zur Auswahl. Die Zahl der Begriffe liegt typischerweise bei einigen tausend.

Für die maximale Anzahl von Begriffen eines Kontextes (O, A, R) existiert eine Abschätzung [18]:

$$|\mathcal{B}(O, A, R)| \leq \frac{3}{2} 2^{\sqrt{|R|+1}} - 1$$

Die experimentell festgestellten Größen liegen allerdings weit unter der durch die Abschätzung gegebenen Höchstgrenze.

Die ermittelten Ergebnisse zeigen, daß die Größe der entstehenden Begriffsverbände und der zeitliche Aufwand für die Verwaltung einer Sammlung von Software-Komponenten in einem angemessenen Verhältnis stehen. Prinzipiell kann die vorgestellte Methode wegen ihrer Komplexität und der Größe der entstehenden Verbände nur für kleine und mittlere Sammlungen verwendet werden.

5.3 Prototyp

Das hier beschriebene Verfahren ist als Prototyp in der funktionalen Sprache Gofer, einem Dialekt von Haskell [6], implementiert. Der Prototyp implementiert die beschriebenen Algorithmen, besitzt allerdings keine spezielle Benutzeroberfläche, die die beschriebenen Verfahren ausnutzen würde. Tabellen sind in dem Prototyp als Red-Black-Trees, Mengen als geordnete Listen implementiert. Der Prototyp ist frei per FTP verfügbar: `ftp.ips.cs.tu-bs.de`, Directory `pub/local/softech/misc`.

6 Andere Ansätze

Das große Interesse an Software-Wiederverwendung hat auch viele Ansätze und Werkzeuge zum Verwalten, Katalogisieren und Suchen von Software-Komponenten hervorgebracht. Die meisten Ansätze lassen sich grob zwei verschiedenen Kategorien zuordnen: zum Ersten den logikbasierten Ansätze. Sie versuchen, den für die Wiederverwendung wichtigen Teil der Semantik einer Komponente formal zu beschreiben. Für die Beschreibung existiert eine Gleichungslogik, die es erlaubt Beschreibungen umzuformen. Der Benutzer kann bei seiner Anfrage eine andere Beschreibung als die von der gesuchten Komponente wählen, da die Gleichungslogik ihre Äquivalenz entscheiden kann. Ziel dieses Ansatzes ist, dem Benutzer eine formale, das heißt mit eindeutiger Semantik versehene, Beschreibung zu geben, mit der er Komponenten suchen kann.

Der zweite generelle Ansatz besteht in der Verwendung allgemeiner *Information Retrieval* Methoden, die spezielle Eigenschaften von Software-Dokumenten nicht verwenden.

Für die Beschreibung der Komponenten existiert keine Gleichungslogik, so daß die Beschreibung einer Komponente und einer Anfrage weitgehende syntaktische Gleichheit bestehen muß, um diese Komponente zu finden. Die Semantik der Beschreibung, die oft aus Schlüsselwörtern besteht, ist oft nicht formal erfaßt, sondern basiert auf der Assoziation des Benutzers mit der Beschreibung. Das vorgestellte Verfahren gehört also zu dieser zweiten Kategorie. Obwohl ein logikbasierter Ansatz aus theoretischer Sicht wegen seiner genauer definierten Semantik interessant ist, wird sein Nutzen in die Praxis von Maarek et. al. und Wood und Sommerville in Frage gestellt [20, 8]. Bei einer formalen Spezifikation müssen Anwender und Klassifikator die formale Sprache beherrschen, was die Hemmschwelle für Wiederverwendung für viele Anwender heraufsetzt. Beim retrievalbasierten Ansatz kann dagegen die Indexierung teilweise automatisiert werden, was gerade bei großen Sammlungen einen Kostenvorteil verspricht, und auch die Formulierung einer Anfrage ist aus Sicht der meisten Anwender einfacher.

6.1 Formal-logische Ansätze

Zur Gruppe des formal-logischen Ansatzes gehören die Arbeiten von Rittri [15, 16], Rollins, Wing und Moormann Zaremski [17, 10] und Fischer et al. [3]. Alle drei Ansätze verwenden Signaturen von Prozeduren als Schlüssel, die beiden letzen Ansätze spezifizieren zusätzlich Vor- und Nachbedingungen. Rittri verwendet eine vollständige Gleichungslogik, die vertauschte Argumente, Currying und Polymorphismus in Signaturen berücksichtigt. Die Gleichungslogik von Rollins und Wing ist nicht vollständig, sondern wendet systematisch eine Reihe von Operationen an, um Anfrage und Komponentenbeschreibung anzugleichen. Vor- und Nachbedingungen werden durch Prolog-Klauseln höherer Ordnung beschrieben, die bei der Suche abgearbeitet werden. Bei Moormann Zaremski und Wing wird die Suche mit Signaturen auf Module ausgedehnt. Sowohl Rittri als auch Rollins, Wing und Moormann Zaremski wenden ihre Verfahren auf neuere funktionale Sprachen mit automatischer Typinferenz an. Fischer et al. verwenden zur Spezifikation von Vor- und Nachbedingungen VDM und zum Test, ob Vor- und Nachbedingungen der Anfrage die einer Komponente implizieren einen Theorembeweiser. Im Gegensatz zu den anderen Ansätzen verwenden sie ihr Verfahren ausdrücklich auch für prozedurale Sprachen.

6.2 Automatische Dokumentanalyse

Das System GURU von Maarek et al. verwendet Methoden des Information Retrieval zur Indexierung von 1100 Dokumenten der Online-Dokumentation eines Unix-Systems [8]. Bei der Indexierung wird der Inhalt eines Dokumentes automatisch nach speziellen Wort-Paaren mit bestimmten statistischen Eigenschaften durchsucht, die dann ein Profil dieses Dokuments darstellen. Dokumente einer Sammlung werden dann an Hand ihrer Profile mittels einer Clusteranalyse in einer Hierarchie angeordnet, die ähnliche Dokumente dicht beeinander stehen läßt. Die Anfrage kann in natürlicher Sprache formuliert werden und wird dann nach dem selben Verfahren wie die Dokumente analysiert. Das Ergebnis der Analyse wird dann zur Erstellung einer nach Ähnlichkeit zur Anfrage sortierten Liste von Dokumenten benutzt. Ausgehend von gefundenen Dokumenten können inhaltlich verwandte Dokument über die Cluster-Hierarchie inspiziert werden. Automatische Indexierung und die einfache Formulierung von Anfragen machen das Verfahren gleichermaßen attraktiv für Aufbau und Benutzung einer Komponenten-Sammlung. Für den Benutzer ist nicht offensichtlich, warum

bestimmte Komponenten ihm präsentiert werden und wie er seine Anfrage umformulieren kann. Ihm bleibt lediglich die Möglichkeit, über die Cluster-Hierarchie geeignetere Komponenten zu finden. Cluster gruppieren Dokumente und definieren eine Distanz zwischen Dokumenten, während Begriffe Dokumente und Attribute gruppieren und zwischen Dokumenten eine Halbordnung besteht, die sogar ein Verband ist. Der Distanzbegriff einer Clusterhierarchie deckt nur einen Aspekt von Ähnlichkeit ab, während in einem Begriffsverband die Dokumente ermittelt werden können, die beliebigen Kriterien genügen.

6.3 Facettierte Klassifikation

Die von Prieto-Diáz entwickelte facettierte Klassifikation weist Komponenten Schlüsselwörter zur Indexierung zu [12, 14, 13]. Jede Facette einer konkreten Indexierung beschreibt einen festgelegten Aspekt der Komponente, wie etwa das von der Komponente manipulierte Objekt, Rückgabewert oder Sprache der Implementierung. Jede Komponente wird mit genau einem Schlüsselwort für jede Facette indexiert, wobei das Schlüsselwort aus einer für die jeweilige Facette festgelegten Menge von möglichen Schlüsselwörtern stammt. Zwischen den Schlüsselwörtern wird zusätzlich manuell ein konzeptioneller Distanzbegriff in Form eines azyklischen Graphen mit markierten Kanten definiert. Eine Anfrage besteht aus einem Vektor von Schlüsselwörtern – eins für jede Facette. Ist eine entsprechend indexierte Komponente Teil der Sammlung wird sie zurückgegeben, sonst wird versucht Schlüsselwörter der Anfrage durch „nahe andere“ zu ersetzen, um damit eine Komponente der Sammlung zu selektieren. Bei einer Anfrage können Schlüsselwörter für Facetten durch Platzhalter ersetzt werden – allerdings nur jeweils für die letzten Facetten des Vektors [12].

Facettierte Klassifikation hat durch die Angabe von Schlüsselwörtern bei Indexierung und Suche viel mit dem hier präsentierten Ansatz gemein. Während bei facettierter Klassifikation die Reihenfolge der Schlüsselwörter relevant ist, ist sie beim hier präsentierten Ansatz unerheblich: dem Benutzer bleiben alle Freiheiten, welchen Aspekt einer Komponente ihn zuerst interessiert. Zusätzlich existiert nur eine Menge von Schlüsselwörtern die er kennen muß, während bei der facettierten Klassifikation eine für jede Facette existiert. Da bei der facettierten Klassifikation die ersten Facetten immer angegeben werden müssen, ist damit auch die Reihenfolge der Aspekte vorgegeben, nach denen eine Komponente ausgewählt werden kann. Der Aufbau eines Distanz-Graphen für die Schlüsselwörter einer Facette ist aufwendig und die dadurch enstehenden Ergebnisse sind für den Benutzer nicht immer plausibel, da ihm dieser Graph verborgen bleibt. Unklar ist, wie bei facettierter Klassifikation der Benutzer bei der Erweiterung oder Umformulierung seiner Anfrage unterstützt wird.

6.4 Methoden der künstlichen Intelligenz

AIRS ist eine System zur Wiederverwendung von Quellcode und ist eine Verallgemeinerung der facettierten Klassifikation [11]. Komponenten werden durch Name-Wert-Paare indexiert; im Gegensatz zur reinen facettierten Klassifikation kann die Anzahl der Paare zwischen Komponenten variieren. Inhaltlich verwandte Mengen von Komponenten können manuell zu *packages* zusammengefaßt werden. Zwischen Komponenten der Sammlung werden zwei verschiedene Beziehungen durch Graphen mit markierten Kanten definiert: Ein Subsumtionsgraph beschreibt, welche Komponenten durch Kombination anderer Komponenten erstellt werden können, ein Abstandsgraph beschreibt, wie groß der Aufwand ist, eine Komponente aus einer anderen durch Modifikation zu gewinnen. Der Distanzbegriff wird

komponentenweise auf *packages* erweitert. Im Unterschied zur facettierten Klassifikation bestehen hier Beziehungen zwischen Komponenten und nicht zwischen Schlüsselwörtern einer Facette. Eine Anfrage besteht aus einer vorgegebenen Menge von Name-Wert-Paaren durch die alle Komponenten selektiert werden, die mindestens die vorgegebenen Paare aufweisen. Weitere Komponenten können über die Subsumtions- und Abstandsbeziehung ausgewählt werden. Gegenüber dem präsentierten Verfahren werden die Komponenten manuell mit sehr viel Struktur versehen. Beim präsentierten Verfahren dagegen sind alle Komponenten zunächst unabhängig von einander – alle existierenden Beziehungen werden vollständig aus der angegebenen Indexierung abgeleitet. Die manuelle Angabe von Beziehungen erlaubt zwar die Sammlung mit einer reichhaltigen Struktur zu versehen, birgt aber die Gefahr von Inkonsistenzen bei Veränderung der Sammlung. Es bleibt unklar, wie der Benutzer bei der Reformulierung seiner Anfrage unterstützt wird.

6.5 Begriffsverbandbasierte Ansätze

Godin et al. schlagen ebenfalls Begriffsverbände zur Navigation in Sammlungen vor, allerdings nicht speziell auf Sammlungen von wiederverwendbaren Software-Komponenten bezogen [5]. Sie beschreiben eine prototypische Implementierung, bei der zur Verschärfung einer Anfrage allerdings nur Schlüsselwörter angeboten werden, die durch direkte Unterbegriffe des aktuellen Begriffes eingeführt werden. Der Benutzer erhält damit also nur eine Teilmenge aller sinnvollen Schlüsselwörter. Der Artikel beschreibt vor allen Dingen Aspekte der Benutzerführung und geht nicht auf algorithmische Aspekte und die Komplexität des Verfahrens ein – eine Abschätzung über die Anwendbarkeit für eine spezielle Anwendung fehlt.

Malischewski präsentiert ein Verfahren zur Auswahl von Werkstoffen mit reellwertigen Attributen auf der Basis von Begriffsverbänden [9]. Das vorgeschlagene Verfahren arbeitet zweistufig: zunächst wird an Hand der an einer Anfrage beteiligten Attribute eine Menge von in Frage kommenden Werkstoffen mit Hilfe eines entsprechenden (statischen) Begriffsverbandes gesucht. Die Anfrage fordert für den gesuchten Werkstoff nicht nur die Existenz eines bestimmter Attribute, sondern auch bestimmte *constraints*, etwa in Form einer Ungleichung. In einem zweiten Schritt wird der Begriffsverband des Kontextes aus augewählten Werkstoffen und den zu erfüllenden constraints berechnet (dynamisch). Dieser Verband wird dem Benutzer graphisch präsentiert und durch Manipulation der Constraints kann der Benutzer seine Anfrage verändern, bis tatsächlich ein passender Werkstoff gefunden wird. Das dargestellte Verfahren ist nicht implementiert und Abschätzungen für den zu erwartenden Aufwand fehlen. Durch seinen reellwertigen Kontext ist es nur schwer mit dem vorgeschlagenene Verfahren zu vergleichen, bietet aber gleichfalls dem Benutzer Hinweise, wie er seine Anfrage geeignet manipulieren kann.

7 Ausblick

Zur weiteren Überprüfung des präsentierten Verfahrens soll es über den Prototypen hinaus implementiert und praktisch erprobt werden. Diese Implementierung wird ihren Schwerpunkt im Gegensatz zum Prototyp in der Benutzerführung haben. Dabei soll dem Benutzer allerdings nicht der Begriffsverband oder Ausschnitte davon graphisch präsentiert werden, sondern eine kontextsensitive Unterstützung, die natürlich auf einer internen Repräsentation des Verbandes beruht. Für die Realisierung der Benutzerführung ist zum einen eine

graphische Oberfläche denkbar, die dem Benutzer in Listen und Fenstern seine selektierten Dokumente, Schlüsselwörter und sinnvollen Erweiterungen seiner Anfrage präsentiert. Zum Zweiten ist auch eine Kommando-orientierte Oberfläche denkbar, die durch eine Vervollständigung für Schlüsselwörter unterstützt wird: beim Eintippen eines Wortes kann der Benutzer dieses Wort durch Druck auf eine spezielle Taste vervollständigen lassen. Das System erkennt anhand des eingegebenen Wort-Präfixes das Wort und vervollständigt es so weit wie möglich. Sind mehrere Vervollständigungen möglich, werden die verschiedenen Alternativen dem Benutzer angezeigt, ebenso wird signalisiert, wenn das eingegebene Wort kein gültiger Präfix ist. Benutzern von Unix-Systemen ist dieses Verhalten als *filename completion* bekannt.

Im konkreten Fall bietet es sich an, die Eingabe von Schlüsselwörtern durch Completions zu unterstützen. Dabei können folgende Situationen unterschieden werden:

- Der Benutzer hat den Präfix eines Schlüsselwortes eingegeben, das Wort aber noch nicht abgeschlossen (etwa durch ein Leerzeichen). Das System kann jetzt an Hand der Liste aller Schlüsselwörter und der aktuellen Position im Verband entscheiden, ob der eingegebene Präfix gültig ist, und welche Vervollständigungen noch möglich sind. Diese werden dann vorgeschlagen oder, solange sie eindeutig sind, unmittelbar ausgeführt.

- Der Benutzer kann die aktuelle, möglicherweise leere, Menge von Schlüsselwörtern durch ein weiteres ergänzen. Sowohl die Menge der selektierten Komponenten, als auch die Menge der noch möglichen Schlüsselwörter können dem Benutzer angezeigt werden.

Die Implementierung soll dann im praktischen Einsatz zur Verwaltung der Online-Dokumentation eines Unix-Systems eingesetzt werden.

Die Verknüpfung von Schlüsselwörtern bei einer Suche ist konjunktiv: selektiert werden alle Dokumente, die die angegebenen Schlüsselwörter mindestens gleichzeitig tragen. Bei Anfrage nach Dokumenten können Negationen und Disjunktionen ebenfalls berücksichtigt werden, allerdings nicht ganz so effizient wie konjunktive Anfragen. Anfragen können disjunktiv verknüpft werden, indem die von den einzelnen Anfragen selektierten Mengen vereinigt werden. Auf gleiche Weise kann Negation durch die Differenzbildung von Mengen implementiert werden. Das Ergebnis einer disjunktive Verknüpfung kann nicht wie eine konjunktive Verknüpfung als Unterverband des Begriffsverbandes dargestellt werden, ergibt sich also nicht alleine durch Verbandsoperationen und kann deswegen nicht so effizient berechnet werden. Für praktische Anwendungen könnte die erhöhte Ausdruckskraft der Anfrage aber wichtiger sein als eine größtmögliche Effizienz, zumal sie sich ja nur bei der tatsächlichen Verwendung von Disjunktionen verringert.

8 Zusammenfassung

Software-Dokumente können zur Suche in einer Sammlung mit Schlüsselwörtern frei indexiert werden. Die Relation aus Dokumenten und Schlüsselwörtern bilden einen formalen Kontext, der mit Hilfe von formaler Begriffsanalyse gewinnbringend zur Suche eingesetzt werden kann. Der Benutzer kann durch Angabe von Schlüsselwörtern Dokumente mit einer eindeutigen Semantik selektieren und wird bei seiner Suche stark kontextsensitiv un-

terstützt. Zu jeder Anfrage kann effizient bestimmt werden, wie diese Anfrage weiter spezialisiert werden kann und welche Dokumente sie bereits selektiert. Die Berechnung einer spezialisierte Anfrage ist inkrementell: die Ergebnisse der alten Anfrage können zur Berechnung die neuen verwendet werden. Dies erlaubt, eine Benutzerschnittstelle mit hoher Interaktivität zu gestalten. Jedes Dokument einer Sammlung wird unabhängig von anderen indexiert und kann somit ohne Gefährdung der Konsistenz aus einer Sammlung entfernt oder ihr hinzugefügt werden.

Für alle bei der Suche relevanten Mengen wurde angegeben, wie sie mit Hilfe des Begriffsverbandes der indexierten Dokumente bestimmt werden können. Zusätzlich wurde für eine Implementierung dieser Operationen eine effiziente Repräsentation des Begriffsverbandes angegeben, die alle Operationen auf einfache Mengen-Operationen zurückführt. Die für die Suche notwendigen Operationen wurden ebenfalls für die vorgeschlagenene Repräsentation dargestellt und in einem frei verfügbaren funktionalen Prototyp verwirklicht.

Bei der Verwendung von Begriffsverbänden zur Suche steht eine effiziente Suche einem erhöhten einmaligem Aufwand zur Berechnung des Begriffsverbandes gegenüber. Durch Experimente wurde belegt, daß dieser Zeit- und Speicheraufwand sich für die Suche in kleineren und mittleren Anwendungen in einem vernünftigem Verhältnis zum Ziel befindet und somit die praktische Anwendbarkeit des vorgeschlagenen Verfahrens gegeben ist.

Ein Vergleich mit anderen Methoden zur Suche in Software-Bibliotheken ergibt die Einordnung des Verfahrens in die Klasse der Information-Retrieval-orientierten Ansätze. Darüberhinaus ist die Untersuchung der Anwendbarkeit von Begriffsanalyse zur Suche von Software-Dokumenten neu.

Literatur

[1] Aït-Kaci, H., Boyer, R., Lincoln, P. & Nasr, R. (1989), 'Efficient Implementation of Lattice Operations', *ACM Transactions on Programming Languages and Systems* **11**(1), 115–146.

[2] Davey, B. A. & Priestley, H. A. (1990), *Introduction to Lattices and Order*, 2nd, Cambridge University Press, Cambridge, GB, Formal Concept Analysis, 221–236.

[3] Fischer, B., Kievernagel, M. & Struckmann, W. (1994), VCR: A VDM-based Software Component Retrieval Tool, Draft, TU Braunschweig, Institut für Programmiersprachen und Informationssysteme, Gaußstraße 17, D-38106 Braunschweig. `ftp.ips.cs.tu-bs.de: /pub/local/softech/drafts/retrieval-with-vdm.ps.gz`.

[4] Ganter, B. (1986), Algorithmen zur Formalen Begriffsanalyse, *in* B. Ganter, R. Wille & K. E. Wolff, Hrsg., 'Beiträge zur Begriffsanalyse', BI Wissenschaftsverlag, Mannheim, pp. 241–254.

[5] Godin, R., Gecsei, J. & Pichet, C. (1989), Design of a Browsing Interface for Information Retrieval, *in* N. J. Belkin & C. J. van Rijsbergen, Eds, '12th Int. SIGIR Conference', ACM, ACM Press, Cambridge, Massachusetts, 32–39.

[6] Hudak, P., Peyton Jones, S. & Wadler, P. (1992), 'Report on the Programming Language Haskell, A Non-Strict Purely Functional Language', *ACM Sigplan Notices*.

[7] Krueger, C. W. (1992), 'Software Reuse', *ACM Computing Surveys* **24**(2), 131–183.

[8] Maarek, Y. S., Berry, D. M. & Kaiser, G. E. (1991), 'An Information Retrieval Approach For Automatically Constructing Software Libraries', *IEEE Transactions on Software Engineering* **SE-17**(8), 800–813.

[9] Malischewski, C. (1992), Ein begriffsverbandsbasiertes Navigationskonzept zur Unterstützung von Recherchen im Eigenschaftsraum eines Werkstoffinformationssystems, Informatik-Bericht 92/3, Institut für Informatik, Clausthal-Zellerfeld.

[10] Moorman Zaremski, A. & Wing, J. M. (1993), Signature Matching: A Key to Reuse, *in* D. Notkin, ed., 'Proc. of the 1st ACM SIGSOFT Symposium on the Foundation of Software Engineering', ACM, ACM Press, Los Angeles, CA, 182–190.

[11] Ostertag, E., Hendler, J., Prieto-Diáz, R. & Braun, C. (1992), 'Computing Similarity in a Reuse Library System: An AI-Based Approach', *ACM Transactions on Programming Languages and Systems* **1**(3), 205–228.

[12] Prieto-Diáz, R. (1987), 'Classifying Software for Reuse', *IEEE Software* **4**(1), 6–16.

[13] Prieto-Diaz, R. (1990), Implementing Faceted Classification for Software Reuse, *in* 'Proceedings of the 12th International Conference on Software Engineering', 300–304.

[14] Prieto-Diáz, R. (1991), 'Implementing Faceted Classification for Software Reuse', *Journal of the ACM* **34**(5), 89–97.

[15] Rittri, M. (1991), 'Using types as search keys in function libraries', *Journal of Functional Programming* **1**(1), 71–89.

[16] Rittri, M. (1992), Retrieving Library Identifiers via Equational Matching of Types, Report 65, Chalmers University of Technology and University of Göteborg, Göteborg, Sweden.

[17] Rollins, E. J. & Wing, J. M. (1991), Specifications as Search Keys for Software Libraries, *in* K. Furukawa, ed., 'Logic Programming: Proc. of the 8th Int. Conference', MIT Press, Paris, France, pp. 173–187.

[18] Schütt, D. (1987), Abschätzung für die Anzahl der Begriffe von Kontexten, Diplomarbeit, Fachbereich Mathematik, TH Darmstadt, Darmstadt, Germany.

[19] Wille, R. (1990), Concept Lattices And Conceptual Knowledge Systems, Preprint 1340, TH Darmstadt, Darmstadt, Germany.

[20] Wood, M. & Sommerville, I. (1988), 'An Information Retrieval System for Software Components', *SIGIR Forum* **22**(3), 11–25.