Concept-Based Component Retrieval

Christian Lindig
Abteilung Softwaretechnologie
Technische Universität Braunschweig
D-38106 Braunschweig
EMail: lindig@ips.cs.tu-bs.de

Abstract

Reusable software components from a library are individually indexed with a set of keywords. To retrieve components the user incrementally specifies a set of keywords that the searched components are required to have. After each step the selected components and the exact set of remaining significant keywords needed to refine the query further are presented to the user. The process ensures that at least one component is found and the user cannot specify conflicting keywords. The efficient computation of retrieved components and significant keywords is based on the precalculated *concepts* of the library, which are natural pairs of component and keyword sets. The concepts form a lattice of super- and subconcepts and are obtained by *formal concept analysis* of the relation over components and keywords. The two main theorems state how to calculate the result of a query and the remaining significant keywords using the concept lattice. An implementation of the proposed approach shows that the user can select components quickly and precisely.

1 Introduction

Formally well founded retrieval approaches are often specific to a small class of programming languages and require trained users [Moorman Zaremski and Wing, 1993; Rittri, 1991; Rollins and Wing, 1991; Kievernagel *et al.*, 1995]. Information retrieval approaches on the other hand are more general and easier to use [Ostertag *et al.*, 1992; Maarek *et al.*, 1991; Prieto-Diaz, 1990] but sometimes lack clear semantics. We present a formally founded and easy-to-use information retrieval approach which permits fast incremental search with strong feedback to the user. For that purpose an initially unstructured collection of components is structured according to its concepts as determined by *formal concept analysis*.

Each component is assigned an arbitrary number of keywords. Formal concept analysis then groups components and keywords for later retrieval to *concepts*, forming a complete lattice of super- and subconcepts. For retrieval the user specifies a query as a set of keywords, selecting all components with at least those keywords assigned. He chooses the keywords incrementally from a list presenting only significant keywords: each chosen keyword refines the query and shrinks the result to a *non empty* set of selected components. That is, the user can specify non-empty component sets only.

2 Example

Figure 1 shows the names of 12 Unix system calls together with a one-line description from their documentation [Sun Microsystems, 1990] and assigned keywords. Formal concept analysis groups names and keywords to *concepts* which are maximal pairs of names and keywords. This can be regarded as an automatic indexing stage, needed to be performed initially and once the collection has changed.

Figure 2 shows an incremental query: initially all keywords may be chosen and all components are selected. After the user has specified the keyword *change* only components indexed with *change* are selected and the set of selectable keywords shrinks to the remaining significant keywords. Calculating the result of a query and the set of remaining significant keywords is efficient and thus permits immediate response even for large collections. The user refines the query by choosing one more keyword from the presented list and thus the number of selected components and remaining keywords decreases, but at least one selected component is guarranteed. After the user has chosen a third keyword exactly one component and no significant keywords remain.

3 Formal Concept Analysis

Formal Concept Analysis was founded by Wille [Wille, 1982; Davey and Priestley, 1990a] and calculates a concept lattice from a binary relation of objects and attributes, called a formal context. The concept lattice allows queries to be processed efficiently.

Definition 1 *A* context *is a triple* $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ *where* $\mathcal{O}$

sys. call	short description	keywords
`chmod`	change mode of file	*change mode permission file*
`chown`	change owner and group of a file	*change owner group file*
`stat`	get file status	*get file status*
`fork`	create a new process	*create new process*
`chdir`	change current working directory	*change directory*
`mkdir`	make a directory file	*create new directory*
`open`	open or create a file for reading or writing	*open create file read write*
`read`	read input	*read file input*
`rmdir`	remove a directory file	*remove directory file*
`write`	write output	*write file output*
`creat`	create a new file	*create new file*
`access`	determine accessibility of file	*check access file*

Figure 1: Unix system calls together with short description and assigned keywords

step	keywords	components	remaining keywords
1	–	all	all
2	*change*	`chdir chmod chown`	*directory file group mode owner permission*
3	*change file*	`chmod chown`	*group mode owner permission*
4	*change file mode*	`chmod`	–

Figure 2: Incremental query

and $\mathcal{A}$ are sets of objects and attributes respectively and $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ is a relation among them.

Keyword indexed components are regarded to be a formal context with components as objects and keywords as attributes. A context can be visualized as a cross table called a context table. Objects from a context share a set of common attributes and vice versa:

Definition 2 *For $O \subseteq \mathcal{O}$ and $A \subseteq \mathcal{A}$ from a formal context $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ define common attributes $\omega(O) \stackrel{\text{def}}{=} \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in \mathcal{R}\}$ and common objects $\alpha(A) \stackrel{\text{def}}{=} \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in \mathcal{R}\}$.*

A *concept* is a pair of objects and attributes, each synonymous with the other:

Definition 3 *A* concept *$c = (O, A)$ of a context $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is a pair where $O \subseteq \mathcal{O}$, $A \subseteq \mathcal{A}$, $\alpha(A) = O$ and $\omega(O) = A$. The* extent *of c is $\pi_o(c) \stackrel{\text{def}}{=} O$ while the* intent *is $\pi_a(c) \stackrel{\text{def}}{=} A$. The set of all concepts of $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is denoted by $B(\mathcal{O}, \mathcal{A}, \mathcal{R})$.*

Concepts are (partially) ordered—a concept's extent includes the extents of its subconcepts and the intent of a concept includes the intents of its superconcepts:

Definition 4 *Concepts $c_1 = (O_1, A_1)$, $c_2 = (O_2, A_2) \in B(\mathcal{O}, \mathcal{A}, \mathcal{R})$ are ordered by the subconcept-relation $\leq$: $c_1 \leq c_2$ iff $O_1 \subseteq O_2$. The structure of B and $\leq$ is denoted by $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$.*

The basic theorem of concept analysis states that the concepts induced by a formal context form a complete lattice. Any number of concepts have one greatest subconcept and one greatest superconcept in common.

Theorem 1 ([Wille, 1982]) *Let $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ be a context. Then $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is a complete lattice, the* concept lattice *of $(\mathcal{O}, \mathcal{A}, \mathcal{R})$. Infimum and suppremum are given as follows: $\bigwedge_{i \in I}(O_i, A_i) = \big(\bigcap_{i \in I} O_i\,,\, \omega\big(\alpha\big(\bigcup_{i \in I} A_i\big)\big)\big)$ and $\bigvee_{i \in I}(O_i, A_i) = \big(\alpha\big(\omega\big(\bigcup_{i \in I} O_i\big)\big),\, \bigcap_{i \in I} A_i\big)$.*

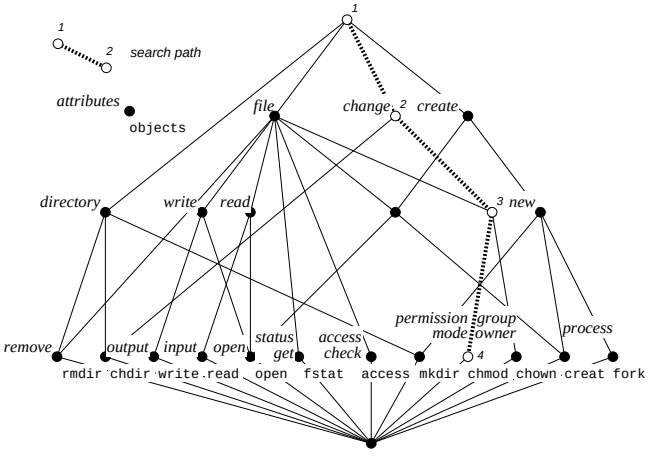

Figure 3: Concept lattice for unix system calls

Figure 3 shows the concept lattice from the example in figure 1. Each step from the query in figure 2 denotes a concept, which together form a path. Instead of the full pairs each concept c is labelled only with objects (system calls) and attributes (keywords) introduced by c. Each attribute is introduced by exactly one concept and passed on to subconcepts; therefore each subconcept of the concept labelled *file* also has *file* in its intent. The same is true for objects, which are passed on to superconcepts—so the exact concepts can be derived from the figure. The concept introducing a given object or attribute is denoted by $\sigma(o)$ and $\mu(a)$ respectively:

Theorem 2 *Let $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ be a concept lattice and let $\sigma(o)$ for $o \in \mathcal{O}$ denote the smallest concept c for which $o \in \pi_o(c)$ holds. Let $\mu(a)$ for $a \in \mathcal{A}$ denote the greatest concept c for which $a \in \pi_a(c)$ holds. These concepts can be expressed as follows: $\sigma(o) = (\alpha(\omega(\{o\})), \omega(\{o\}))$ and $\mu(a) = (\alpha(\{a\}), \omega(\alpha(\{a\})))$.*

There exist several algorithms to compute all concepts from a context $(\mathcal{O}, \mathcal{A}, \mathcal{R})$, especially the one by Ganter [Ganter, 1986]. Because a concept lattice can consist of 2^n concepts, in the worst case, with $n = min(|\mathcal{O}|, |\mathcal{A}|)$, their complexity is exponential. However, in our experience, the number of concepts typically grows linearly with the number of objects. Ganter's algorithm has time complexity $O(n \cdot |\mathcal{A}| \cdot |\mathcal{O}|^2)$ where n is the number of concepts of $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R}))$ [Ganter, 1986] and thus the typical complexity is polynomial.

4 Concept Based Retrieval

Keyword indexed components form a context which induces a concept lattice. The lattice is used to perform queries efficiently and to give optimal feedback to the user. A query is a set of keywords selecting all objects assigned at least those keywords:

Definition 5 (Query) *A set $A \subseteq \mathcal{A}$ is a* query *to a concept lattice $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$. A component $o \in \mathcal{O}$ satisfies a query, iff $\omega(o) \supseteq A$ holds. The set of all components satisfying a query is called* result *and is denoted by $[\![A]\!] \stackrel{\text{def}}{=} \{o \mid o \in \mathcal{O}, \omega(o) \supseteq A\}$.*

Each keyword from a query is introduced by a concept. The extent of the infimum of all these concepts constitutes the result of the query: all objects of concepts smaller than the infimum are also part of the result.

Theorem 3 *Let $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ be a concept lattice and let $A \subseteq \mathcal{A}$ be a query, then $[\![A]\!] = \pi_o\Big(\bigwedge_{a \in A} \mu(a)\Big)$ holds.*

Once the concept lattice for a collection has been calculated, the result for any query can be calculated efficiently using theorem 3. After a user has specified a query, how can he *refine* it by adding another keyword to the query but ensuring the new result is not empty? The set of keywwords he can choose from is called the set of significant keywords:

Definition 6 (Significant Keywords) *Let $A \subseteq \mathcal{A}$ be a query to $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$. The set of* significant keywords *is denoted by $\langle\langle A \rangle\rangle \stackrel{\text{def}}{=} \{a \in \mathcal{A} \mid \emptyset \subset [\![A \cup \{a\}]\!] \subset [\![A]\!]\}$*

The set of significant keywords can be also computed efficiently using theorem 4.

Theorem 4 *Let $A \subseteq \mathcal{A}$ be a query to $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$, then for the significant keywords which refine A the following holds:*

$$\langle\langle A \rangle\rangle = \Big(\bigcup_{o \in [\![A]\!]} \omega(o) \Big) \setminus \pi_a\Big(\bigwedge_{a \in A} \mu(a) \Big)$$

After the user has refined his query A by adding $a \in \langle\langle A \rangle\rangle$ the new result $[\![A \cup \{a\}]\!]$ is determined incrementally by computing one infimum: $[\![A \cup \{a\}]\!] = \pi_o(c_{[\![A]\!]} \wedge \mu(a))$ where $c_{[\![A]\!]} = \bigwedge_{a \in A} \mu(a)$ is the infimum from the previous computation of $[\![A]\!]$.

5 Experiments

The presented approach has been implemented as a prototype with a graphical user interface (figure 4) to manage a collection of 1658 documents of a Unix online documentation. It permits fast incremental queries, as shown, and lets the user view the selected documents. The documents were indexed with keywords from a set of 92 keywords; the resulting concept lattice consists of 714 concepts and its initial computation takes 150 seconds on a SPARCstation ELC.

For the collection of documents we investigated the distribution of selected documents after two query steps and the distribution of remaining significant keywords. In 52% of all cases the number of selected documents is less than 6 and the number of remaining keywords in 68% of all cases is less than 2. Thus each step narrows the search spaces significantly and so the user is led rapidly to the searched documents.

We investigated systematically the time- and space-complexity because, in the worst case, computing all concepts from a context is exponential in time and space. Randomly generated contexts with up to 2000 objects were used to calculate the corresponding concept lattices. Computing time ranged from 0.1 to 1000 seconds, while the number of concepts varied from approx. 50 to 8000 which is far below the upper bound. The proposed approach is thus well suited for at least small and middle sized collections.

6 Conclusions

Keyword indexed components induce a concept lattice which permits a fast, exact and incremental retrieval of components. Queries have clear semantics and thus the obtained results are comprehensible to the user. Because each component is indexed independently of others the collection of components is easy to maintain. The recalculation of the concept lattice, once the collection has

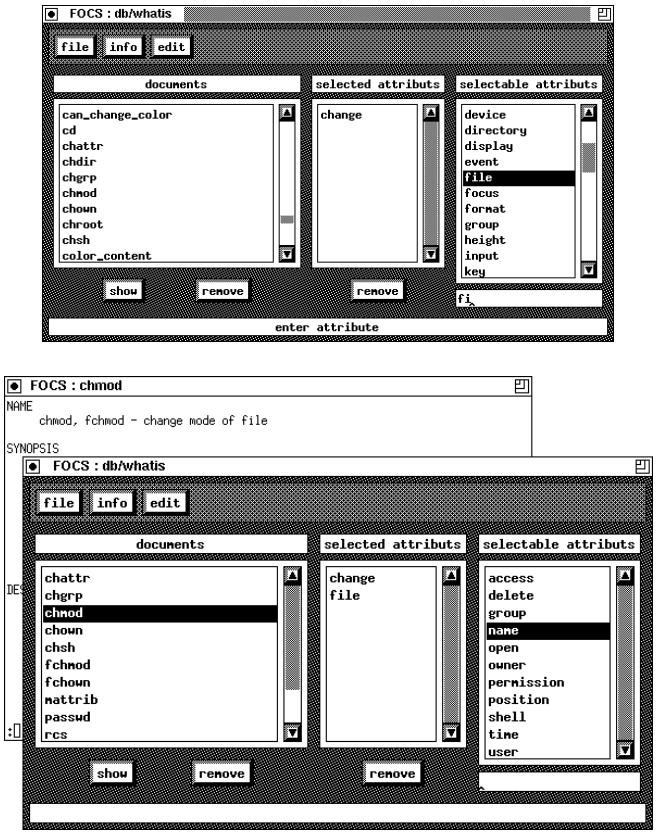

Figure 4: Selecting all documents indexed with *change* and *file*.

changed, has typically polynomial effort as our experiences have shown. Our prototype implemented to manage a collection of documents is promising: a user can obtain documents with a few mouse clicks. This impression is supported by the distribution we found for selected components and significant keywords. Moreover, the concept-based approach is not specific to online documentation but also permits management of heterogeneous and multi-media collections. So concept analysis has again appeared useful in software engineering [Krone and Snelting, 1994].

Acknowledgements: Peter Dettmer implemented the prototype. Bernd Fischer, Franz-Josef Grosch and Gregor Snelting provided valuable comments of earlier versions of this paper.

A Proofs

A.1 Lemma

Let $\mathcal{B}(\mathcal{O},\mathcal{A},\mathcal{R})$ be a concept lattice. The pair of maps α and ω form a Galois connection, that is for $O_1, O_2 \subseteq \mathcal{O}$ and $A_1, A_2 \subseteq \mathcal{A}$ the following holds:

1. $O_1 \subseteq O_2 \;\Rightarrow\; \omega(O_1) \supseteq \omega(O_2)$
2. $A_1 \subseteq A_2 \;\Rightarrow\; \alpha(A_1) \supseteq \alpha(A_2)$
3. $O_1 \subseteq \alpha(\omega(O_1))$ and $\omega(O_1) = \omega(\alpha(\omega(O_1)))$
4. $A_1 \subseteq \omega(\alpha(A_1))$ and $\alpha(A_1) = \alpha(\omega(\alpha(A_1)))$

Proof: [Birkhoff, 1967], pp. 122.

A.2 Theorem 2

Proof: [Davey and Priestley, 1990b], pp 221–236.

A.3 Theorem 3

$$\begin{aligned}\pi_o\Big(\bigwedge_{a\in A}\mu(a)\Big) &= \pi_o\Big(\bigwedge_{a\in A}(\alpha(a),\omega(\alpha(a)))\Big)\\ &= \bigcap_{a\in A}\alpha(a) = \bigcap_{a\in A}\{o\in\mathcal{O}\mid(o,a)\in\mathcal{R}\}\\ &= \{o\in\mathcal{O}\mid\forall a\in A:(o,a)\in\mathcal{R}\}\\ &= \{o\in\mathcal{O}\mid\omega(o)\supseteq A\} = [\![A]\!]\end{aligned}$$

A.4 Theorem 4

1. Let be $a_1 \in \langle\!\langle A\rangle\!\rangle$. We prove that (1) $a_1 \in \bigcup_{o\in[\![A]\!]}\omega(o)$ and (2) $a_1 \notin \pi_a(\bigwedge_{a\in A}\mu(a))$ holds.

   (1) Let be $o_1 \in [\![A\cup\{a_1\}]\!]$ and consider:

$$\begin{aligned}o_1 \in [\![A\cup\{a_1\}]\!] &= \pi_o\Big(\bigwedge_{a\in A\cup\{a_1\}}\mu(a)\Big)\\ &= \bigcap_{a\in A\cup\{a_1\}}\alpha(a) = \bigcap_{a\in A}\alpha(a)\cap\alpha(a_1)\end{aligned}$$

   From this $o_1 \in \alpha(a_1) \Leftrightarrow (o_1,a_1)\in\mathcal{R} \Leftrightarrow a_1\in\omega(o_1)$ follows. Because $o_1 \in \bigcap_{a\in A}\alpha(a) = [\![A]\!]$ holds, a_1 is member of the union over all $\omega(o)$:

$$a_1 \in \bigcup_{o\in[\![A]\!]}\omega(o)$$

   (2) Assume $a_1 \in \pi_a(\bigwedge_{a\in A}\mu(a))$. ω maps the object set of a concept to the set of common attributes so the assumption can be expressed as:

$$\{a_1\} \subseteq \pi_a\Big(\bigwedge_{a\in A}\mu(a)\Big) = \omega\Big(\pi_o\Big(\bigwedge_{a\in A}\mu(a)\Big)\Big)$$

   Because α and ω form a Galois conncetion we can show, that $\alpha(a_1)$ is part of $[\![A]\!]$:

$$\begin{aligned}\alpha(a_1) &\supseteq \alpha\Big(\omega\Big(\pi_o\Big(\bigwedge_{a\in A}\mu(a)\Big)\Big)\Big)\\ &= \pi_o\Big(\bigwedge_{a\in A}\mu(a)\Big) = \bigcap_{a\in A}\alpha(a) = [\![A]\!]\end{aligned}$$

   That is, $\alpha(a_1)$ is superset of $\bigcap_{a\in A}\alpha(a)$ and thus for $[\![A\cup\{a_1\}]\!]$ follows:

$$[\![A \cup \{a_1\}]\!] = \bigcap_{a \in A} \alpha(a) \cap \alpha(a_1) = \bigcap_{a \in A} \alpha(a) = [\![A]\!]$$

This contradicts $[\![A \cup \{a_1\}]\!] \subset [\![A]\!]$ and thus $a_1 \notin \pi_a(\bigwedge_{a \in A} \mu(a))$ must hold.

2. Let be $a_1 \in \bigcup_{o \in [\![A]\!]} \omega(o)$ and let be $a_1 \notin \pi_a(\bigwedge_{a \in A} \mu(a))$. We prove (1) $[\![A \cup \{a_1\}]\!] \neq \emptyset$, (2) $o \in [\![A \cup \{a_1\}]\!] \Rightarrow o \in [\![A]\!]$ (that is $[\![A \cup \{a_1\}]\!] \subseteq [\![A]\!]$), (3) $[\![A]\!] \neq [\![A \cup \{a_1\}]\!]$

   (1) From $a_1 \in \bigcup_{o \in [\![A]\!]} \omega(o)$ follows, that $\exists o_1 \in [\![A]\!] : a_1 \in \omega(o_1)$ holds. That is $(o_1, a_1) \in \mathcal{R}$ and thus $o_1 \in \alpha(a_1)$. Consider now $[\![A \cup \{a_1\}]\!]$:

$$\begin{aligned} [\![A \cup \{a_1\}]\!] &= \bigwedge_{a \in A \cup \{a_1\}} \mu(a) = \bigcap_{a \in A \cup \{a_1\}} \alpha(a) \\ &= \bigcap_{a \in A} \alpha(a) \cap \alpha(a_1) \supseteq \{o_1\} \neq \emptyset \end{aligned}$$

   (2) Let be $o_1 \in [\![A \cup \{a_1\}]\!]$. From $[\![A \cup \{a_1\}]\!] = \bigcap_{a \in A} \alpha(a) \cap \alpha(a_1)$ follows that $o_1 \in \alpha(a_1)$ and $o_1 \in \bigcap_{a \in A} \alpha(a) = [\![A]\!]$. Thus $[\![A \cup \{a_1\}]\!] \subseteq [\![A]\!]$ holds.

   (3) Assume $[\![A \cup \{a_1\}]\!] = [\![A]\!]$. That is:

$$\bigcap_{a \in A} \alpha(a) \cap \alpha(a_1) = \bigcap_{a \in A} \alpha(a)$$

   From this $\alpha(a_1) \supseteq \bigcap_{a \in A} \alpha(a)$ follows. Because α and ω form a Galois conncetion applying ω is anti-monotone: $\omega(\alpha(a_1)) \subseteq \omega\Big(\bigcap_{a \in A} \alpha(a)\Big)$

   Left and right hand side are both sets of attribut of concepts and thus can be reformulated:

$$\{a_1\} \subseteq \pi_a(\mu(a_1)) \subseteq \pi_a\Big(\bigwedge_{a \in A} \mu(a)\Big)$$

   This contradicts $a_1 \notin \pi_a(\bigwedge_{a \in A} \mu(a))$.

References

[Birkhoff, 1967] Garret Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, 2rd edition, 1967.

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

[Davey and Priestley, 1990b] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, GB, 2nd edition, 1990.

[Ganter, 1986] Bernhard Ganter. Algorithmen zur formalen Begriffsanalyse. In B. Ganter, R. Wille, and K. E. Wolff, editors, *Beiträge zur Begriffsanalyse*, pages 241–254. BI Wissenschaftsverlag, Mannheim, 1986.

[Kievernagel *et al.*, 1995] Matthias Kievernagel, Bernd Fischer, and Werner Struckmann. VCR: A VDM-based software component retrieval tool. In *ICSE-17 Workshop on* Formal Methods Appl. in SW Eng. Practice, Seattle, USA, 1995. To appear.

[Krone and Snelting, 1994] Maren Krone and Gregor Snelting. On the inference of configuration structures from source code. In *Proc. 16th International Conference on Software Engineering*, pages 49–57, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[Maarek *et al.*, 1991] Yoëlle S. Maarek, Daniel M. Berry, and Gail E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, SE-17(8):800–813, 1991.

[Moorman Zaremski and Wing, 1993] Amy Moorman Zaremski and Jeanette M. Wing. Signature matching: A key to reuse. In David Notkin, editor, *Proc. of the 1st ACM SIGSOFT Symposium on the Foundation of Software Engineering*, pages 182–190, Los Angeles, CA, December 1993. ACM Press.

[Ostertag *et al.*, 1992] Eduardo Ostertag, James Hendler, Rubén Prieto-Diáz, and Christine Braun. Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions on Programming Languages and Systems*, 1(3):205–228, 1992.

[Prieto-Diaz, 1990] R. Prieto-Diaz. Implementing Faceted Classification for Software Reuse. In *Proceedings of the 12th International Conference on Software Engineering*, pages 300–304, March 1990.

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

[Rollins and Wing, 1991] E. J. Rollins and J. M. Wing. Specifications as search keys for software libraries. In Koichi Furukawa, editor, *Logic Programming: Proc. of the 8th Int. Conference*, pages 173–187, Paris, France, June 1991. MIT Press.

[Sun Microsystems, 1990] Sun Microsystems. *SunOS Reference Manual*. Sun Microsystems, Inc., SunOS 4.1.1, 1990.

[Wille, 1982] Rudolf Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. Reidel, 1982.