SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 28(8), 799–818 (10 JULY 1998)

A Domain-Independent Software Reuse Framework Based on a Hierarchical Thesaurus

hsien-chou liao, ming-feng chen and feng-jian wang
Department of Computer Science and Information Engineering, National Chiao-Tung University, 1001 TaHsueh Road, Hsinchu 30050, Taiwan, R.O.C.
(email: {hcliao,mfchen,fjwang}@csie.nctu.edu.tw)

SUMMARY

Software reuse is an effective way to improve software productivity and quality. Software libraries are getting bigger, while most of them, such as those for object-oriented languages, use simple but somewhat ineffective classification methods. These libraries typically provide search aids for novices, but not for experts. They are not flexible enough to adequately serve users with different abilities. In this paper, a Software Reuse Framework (SRF) for overcoming these drawbacks is proposed. Based on a built-in hierarchical thesaurus, the classification process can be made semi-automatic. SRF is a domain-independent framework that can be adapted to various libraries. SRF also provides four search levels to serve users with different skill levels. Two sample SRF systems* are implemented, one for Smalltalk-80 and the other for MFC 4.0 (Microsoft® Foundation Class) libraries. © 1998 John Wiley & Sons, Ltd.

key words: software reuse; faceted scheme; hierarchical thesaurus

INTRODUCTION

In the past 30 years, software productivity has been growing steadily. Only 15 per cent of existing software programs are domain-specific, 85 per cent are general in nature, i.e. up to 85 per cent of existing software programs have reuse potential in other domains.[1,2] During software development, the process of software reuse can be classified into four phases: *abstraction*, *selection*, *specialization*, and *integration*.[3] The selection phase enables users to locate desired components. Library classification and search processes facilitate the location of components. Search techniques for component classification and search can be divided into the following four categories: *keyword search*, *full-text retrieval*, *structured classification schema*, and *hypertext*.[4] Many approaches to improving classification and search processes have been proposed,[5–9] but most of them have two shortcomings: (1) they usually are labor-intensive in the classification process and cannot be used for different libraries; and (2) they provide search aids for novices but not for experts.

* The sample systems can be accessed from the anonymous FTP site ‘dssl.csie.nctu.edu.tw’ under the directories ‘/download/SRF/Smalltalk’ and ‘/download/SRF/MFC’.

CCC 0038–0644/98/080799–20$17.50
© 1998 John Wiley & Sons, Ltd.
Received 15 July 1996
Revised and Accepted 19 February 1998

The faceted approach is a reuse approach that is widely accepted.[10,11] In the faceted scheme, a thesaurus provides vocabulary control, and a conceptual distance graph is used to evaluate the similarities between terminology of facets. But constructing a thesaurus and a conceptual distance graph is labor-intensive.[12] The conceptual distances are different for novices and for experts; hence, the evaluated result based on the conceptual distance graph with a single level may not be suitable for both novices and experts.

In this paper, a domain-independent Software Reuse Framework (SRF) is proposed. SRF is based on a faceted scheme modified from the one presented in Prieto-Diaz and Freeman.[10] In SRF, an existing thesaurus is transformed into an *hierarchical thesaurus* with a four-level structure. This hierarchical thesaurus incorportes the thesaurus and conceptual distance graph of the original faceted scheme. SRF also provides a semi-automatic classification process based on the hierarchical thesaurus, and it enables *synonymy generation*, which operates on the hierarchical thesaurus to provide a similarity-based search mechanism with four distinct search levels to satisfy users with different ability levels who use the library. The synonyms of words in queries are collected via synonymy generation. The *use frequencies* of words represent their similarities to concepts, and are accumulated via user feedback during searches. The *synonymity weights* between words and synonyms, derived from use frequencies, are the bases upon which SRF computes *similarity degrees* between queries and software components. Finally, the software components returned during searches are listed according to their similarity degrees to reduce the need for further searching. SRF is domain-independent and can be used for different libraries. Two prototypes have been implemented in MS-Windows to demonstrate SRF characteristics.

AN OVERVIEW OF SOFTWARE REUSE FRAMEWORK—SRF

SRF is proposed to meet the following requirements:

1. *Efficiency*: classification should be semi-automatic at least, and should be practical for use with different libraries.
2. *Serves users with multiple skill levels*: users may be novices, moderately skilled, or even experts. Novices usually need search processes that provide all possible components. Conversely, experts need only search mechanisms that provide specific components. Search processes must be usable not only by novices, but also by experts as well.

SRF contains six elements: a *hierarchical thesaurus*, a *classification catalog*, a *special-word dictionary*, *thesaurus pre-processing* process, *classification* process, and *search* process. The SRF data-flow diagram is shown in Figure 1.

As Figure 1 shows, a thesaurus first undergoes *thesaurus pre-processing*, during which plain text from the input thesaurus is analysed and transformed into a *hierarchical thesaurus*. The *special-word dictionary* is used to record all resulting special words and their corresponding words in the *hierarchical thesaurus*. Special words are those that do not exist in the hierarchical thesaurus, but are used in the classification process: they include jargon, proper nouns, abbreviations, and so on. A corresponding weight represents the distance between a special word and its corresponding word. The *classification* process uses the hierarchical thesaurus and special-word dictionary to generate facet descriptors for software components. A

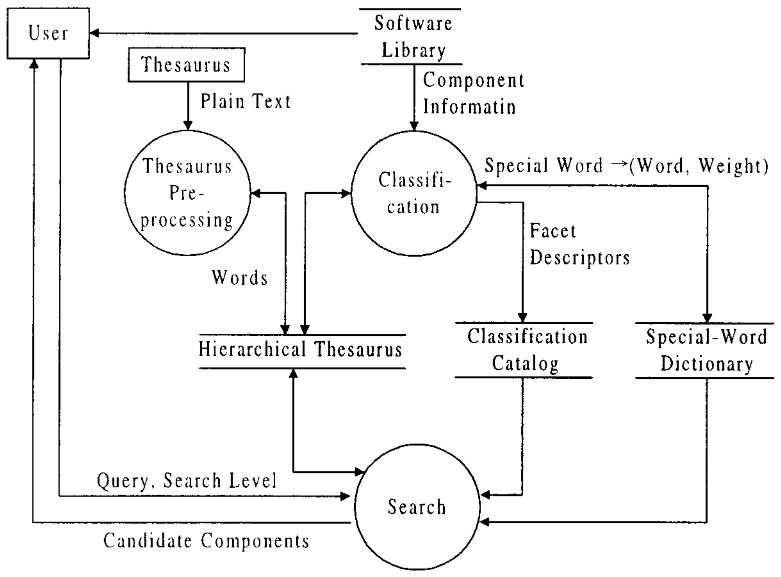


Figure 1. SRF data-flow diagram

facet descriptor is a component index. All facet descriptors are stored in the *classification catalog.*

When a user queries the *Search* process, SRF examines the contents of the classification catalog, hierarchical thesaurus, and special-word dictionary to compute similarity degrees between the query and candidate components. The candidate components are then listed according to their similarity degrees.

In SRF, the hierarchical thesaurus is the key element that enables SRF to satisfy the above classification and search-process requirements. The purpose of classification is to acquire a set of words that describe the features of a component. Classification is conducted in three steps: *Acquisition, Analysis*, and *Clarification*. Utilization of the hierarchical thesaurus in the classification process requires the intervention of domain experts, mainly during clarification. This classification process can be made semi-automatic, as described in the section on ‘The SRF Classification Process’ below.

In the search process, a user enters a query and selects a specific search level. The search levels correspond to the four-level structure of the hierarchical thesaurus and affect the number of candidate components returned. Users with different levels of library experience can search on different search levels. Thus, SRF satisfies users with different skill levels.

THE HIERARCHICAL THESAURUS

Construction

A thesaurus usually abstracts sets of categories from the meanings of selected words. Each category contains a group of synonyms. Synonyms in categories are listed according to their parts of speech: noun, verb, adjective, and adverb. Synonyms of the same type are further grouped according to major semantic definitions to

Table I. An example of a thesaurus listing

#298.	**Relation**
N.	relation, bearing, reference, connection, concern, correlation, analogy, similarity, … comparison, ratio, proportion, link, tie
V.	regard, concern, touch, affect, interest connect, associate, link
Adj.	relative, correlative, cognate related, connected, implicated, associated, affiliated approximating, proportional, proportionate, allusive, comparable like, relevant, applicable
Adv.	relatively thereof, about, concerning, anent, whereas

form a collection of paragraphs. An example of a thesaurus listing is shown in Table I.

The plain text shown in Table I can be transformed into the hierarchical structure shown in Figure 2.

The structure shown in Figure 2 is called a *hierarchical thesaurus*; it has four levels. The levels from top to bottom are, respectively, category, part of speech, major definition, and synonym. Interior nodes represent abstract concepts and leaves represent words. An abstract representation of such a hierarchy is shown in Figure 3.

Level 4 is called the *category* level; it contains all categories in the thesaurus. Level 3 is called the *cluster* level and each category leads to four clusters. The symbols, *N*, *V*, *Adj*, and *Adv* on this level represent noun, verb, adjective, and adverb clusters, respectively. Level 2 is called the *notional* level; on it, each notion is a refinement of a node on Level 3. Level 1 is called the *identity* level and contains all words in the thesaurus.

Synonymy generation

Synonymy generation aims to find synonyms in the hierarchical thesaurus that have common ancestor nodes (i.e. abstract concepts) with input words. It is carried

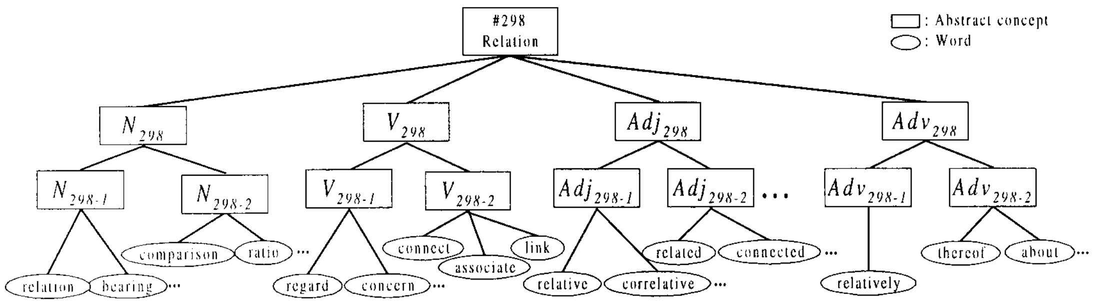


Figure 2. A hierarchical example of a thesaurus listing

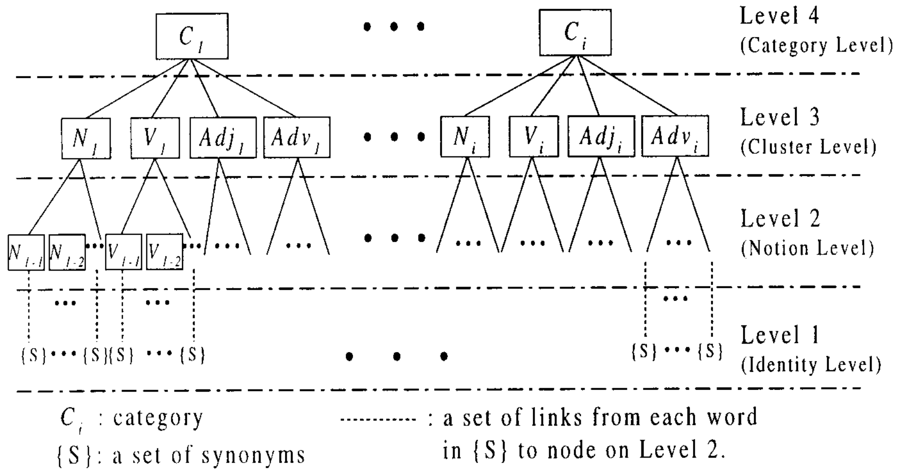

C_1
C_i
Level 4 (Category Level)
N_1 V_1 Adj_1 Adv_1
N_i V_i Adj_i Adv_i
Level 3 (Cluster Level)
$N_{1\text{-}1}$ $N_{1\text{-}2}$ $V_{1\text{-}1}$ $V_{1\text{-}2}$
Level 2 (Notion Level)
{S}•••{S}{S} •••{S}
{S}•••{S}
Level 1 (Identity Level)
C_i : category
{S}: a set of synonyms
: a set of links from each word in {S} to node on Level 2.

Figure 3. An abstract representation of a hierarchical thesaurus

out in two steps: *concept classification* and *synonym collection*. Concept classification means obtaining all concepts on one level for a word. Synonym collection means returning all (descendent) leaves of these concepts. A word that has more meanings usually has more synonyms. Let a synonymy generation work for word **W** on Level **SL**. The synonym generation proceeds as follows:

1. *Concept classification*: collects the set of **W** concepts on **SL**. The result is called the *concept set*, denoted as **CS (W, SL)**:

   CS(W, SL) = {**n** | **n** is an ancestor node of **W** on **SL**.}

2. *Synonym collection*: collects the leaves (i.e. words) that have at least one concept in **CS(W, SL)**. The result is called the *synonymy set*, denoted as **SS(W, SL)**:

   SS(W, SL) = {**w** | **w** is a descendant leaf of **N**, **N** $\in$ **CS(W, SL.}**

The number of synonyms in **SS(W, SL)** is affected by the **SL** set. For any given word **W**, the number of **SS(W, SL)** synonyms is reduced level by level (from Level 4 to 1). This allows SRF to meet the requirement of having *distinct search processes corresponding for users with different skill levels.* SRF thus provides four search levels with respect to the levels in the hierarchical thesaurus. The synonomy generation of a word **W** in a query on Level 4 can return at most **SS(W, SL)**. Thus, a query on Level 4 may return most related candidate components, but queries on Level 1 are thought to use no synonymy generation and return only component(s) containing query words as their keywords, i.e. those that match exactly. The following example shows synonymy generation on Levels 3 and 2.

Let the input word be the *link* in Table I, and let the synonymy generation on Level 3 explore the synonyms that have concepts similar to *link*. Concept classification collects the concepts N_{298} and V_{298} as shown in Figure 2 and synonym collection gets the synonyms: *relation, bearing, reference, …, proportion, link*, and *tie* from N_{298}, and *regard, concern, touch, affect, interest, connect, associate*, and *link* from V_{298}. If the synonymy generation is on Level 2, concept classification collects the concepts $N_{298\text{-}2}$ and $V_{298\text{-}2}$, and synonym collection gets synonyms *comparison, relation, proportion, link*, and *tie* from $N_{298\text{-}2}$, and *connect, associate*, and *link* from $V_{298\text{-}2}$. These synonyms are refinements of the noun and verbal concepts of *link*. Using Level 2 to explore *link* yields fewer but closer synonyms than using Level 3.

SRF allows users to select any level during search. Users may select the ones they need according to how familiar they are with the library. Generally, users who are unfamiliar with the search domain, so-called novices, may choose higher levels (Levels 3 or 4) to get more candidate components. Conversely, users experienced with the domain, so-called experts, may choose lower levels (Levels 1 or 2) to get fewer but more closely related candidate components. When software components are returned by search processes, those more similar to query words are listed first to reduce the need for further searching. Thus, SRF is designed to evaluate the *similarity degrees* between queries and software components. However, software components are described by sets of words in the classification process, and each word may belong to more than one concept. To evaluate the similarities between query words and software components, SRF must first evaluate the similarities between words and concepts. *Use frequency* and *synonymity weight* are proposed to compute these similarities, and are discussed in the next section.

Use frequency and synonymity weight

An SRF query can be viewed as a search for components with one or more desired concepts. The keywords of software components have their own concepts. If a concept derived from query words is also a concept of a software component's keyword, the software component may be the one needed. In other words, this concept is common to both the query words and the keywords of the selected component. Therefore, the common concepts may be the best candidates to represent the real ideas users want to express in their queries. Let each descendent leaf (word) of a concept be associated with a value determined by the *frequency* that the leaf is used as a query word, and the concept is a concept of the selected component's keywords. The software components selected by means of the concept having descendent query words with higher frequencies may be more desirable.

Words can be used for different concepts. A word thus has different use frequencies for each ancestor concept. Let the use frequency of a word **W** representing a concept **C** be denoted as **UF(W, C)**. **UF(W, C)** is added each time a query word **W** matches a selected component's keyword. Given two concepts $\mathbf{C_1}$ and $\mathbf{C_2}$, **W** more closely matches $\mathbf{C_1}$ if $\mathbf{UF(W, C_1)}$ is larger than $\mathbf{UF(W, C_2)}$. Similarly, $\mathbf{W_1}$ more closely matches **C** than $\mathbf{W_2}$ if $\mathbf{UF(W_1, C)}$ is larger than $\mathbf{UF(W_2, C)}$.

As mentioned above, synonymy generation is important to SRF's search process. In keyword-based search tools, components are returned only when queries (partially) match component keywords. When synonymy generation is used, components can be returned when one of their keywords matches one synonym of a query word. Desired components can thus be returned even when the query words are not their keywords. Synonymy generation expands the search scope, and queries may return

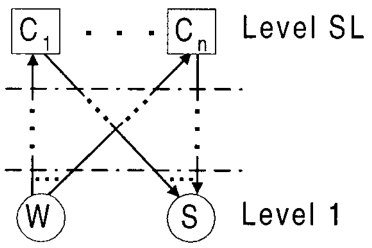


Figure 4. A synonymy generation graph for W and S

large numbers of candidate components, which may inconvenience users. Computing the similarity degrees between returned components and queries to order these components can reduce user effort. SRF uses *synonymity weight* to represent the degree of closeness between a word and its synonym: For a query word, synonyms having the larger synonymity weights are closer.

When the synonyms of a word are collected via a set of concepts, the closeness between the word and these concepts are usually different. One possible reason is that their use frequencies are different. If a concept of high use frequency is selected during synonymy generation, the synonymity weight of the corresponding synonym is considered large. The synomymity weight is defined as follows: Suppose a synonym `S` of a word `W` is collected via synonymy generation on Level `SL, W` and `S` has `n(n≥1)` common concepts, C_k for `1≤k≤n`, the synonymity weight is denoted as `SW(W, S, SL)`. A graph showing synonymy generation between `W` and `S` is given in Figure 4.

Let `UF(W, C`$_k$`)` be `W`'s `k`th concept on Level `SL`, 1≤`k`≤`n`, and `SW(W, S, SL)` be the maximum of `UF(W,C`$_k$`)`, 1≤`k`≤`n`. When `W` is the same as `S, SW(W, S, SL)` is defined as MAX_{SW}, a constant representing the maximum synonymity weight. The equation for `SW(W, S, SL)` is shown below:

$$\mathbf{SW(W, S, SL)} = \begin{cases} \max\limits_{1\le k\le n} [UF(W,C_k)] & \text{if } W \neq S \text{ and } C_k \text{ is a common concept on Level SL for W and S} \\ MAX_{SW} & \text{otherwise} \end{cases} \qquad (1)$$

For example, let `W` be the *link* and `S` be the *concern* in Table I. Figure 5 shows

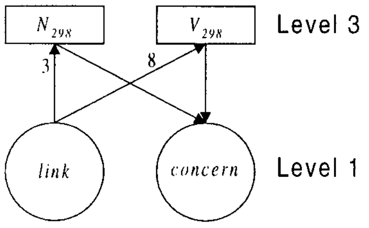


Figure 5. The synonymy generation for link

the synonymy generation for *link* on **`Level 3`**. *link* has two common concepts with concern on **`Level 3`**, and thus has two use frequencies, **`UF`**(*link*, N_{298}) and **`UF`**(*link*, V_{298}). Let the corresponding frequencies be 3 and 8, and labeled on the links from *link* to the common concepts. The synonymity weight **`SW`** (*link*, *concern*, **`3`**) equals 8 from Eq. (1).

THE SRF CLASSIFICATION PROCESS

SRF faceted scheme

SRF uses a *faceted scheme*, adapted from Prieto-Diaz and Freeman,[10] to classify software components. It includes three parts: a hierarchical thesaurus, a special-word dictionary, and a classification catalog. The hierarchical thesaurus was introduced above. The special-word dictionary contains a set of *statements*, each of which contains three tuples:

$$\mathbf{SP} \rightarrow (\mathbf{CWD}, \mathbf{CWT})$$

where **`SP`** represents special words not found in the hierarchical thesaurus but used in the application domain, such as jargon, proper nouns, abbreviations, and so on; **`CWD`** is a word in the hierarchical thesaurus representing **`SP`**; and **`CWT`** describes how closely related **`CWD`** is to **`SP`**. **`CWT`** is a value between 0 and 1, and **`CWT`** = 1 means that **`CWD`** and **`SP`** have the same meaning. An **`SP`** is usually an abbreviation of some keywords or phrases and may occur in more than one statement, i.e. it may be described by more than one **`CWD`**. For example, the **`SP`** *OS* may be described by two **`CWD`**s, *operation* and *system*. Thus, the special-word dictionary contains the statements, *OS* → (*operation*, 1) and *OS* → (*system*, 1).

The classification catalog stores all the *facet descriptors* for software components, where a facet descriptor is a component index used for comparison during the search process. Each facet descriptor consists of a set of *facet-value pairs*. A component's facet-value pair represents some component characteristics. The pair consists of a *facet value* and a *facet-value weight*. A facet value is a word in either the hierarchical thesaurus or the special-word dictionary. The facet-value weight associated with a component represents how well the facet value describes the component. An abstract representation of the faceted scheme in SRF is shown in Figure 6. In addition to

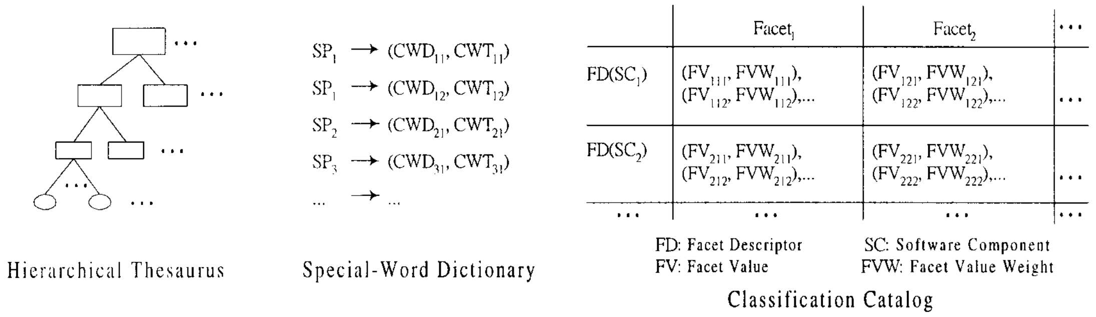


Figure 6. An abstract representation of the SRF faceted scheme

the hierarchical thesaurus and special-word dictionary, the table in Figure 6 also represents the classification catalog, in which **SC** denotes a software component, **FD(SC)** denotes the facet descriptor for **SC, FV** denotes a facet value, **FVW** denotes a facet-value weight, and **(FV, FVW)** denotes a facet-value pair. The set of facet-value pairs of a component $\mathbf{SC}_i$ for the facet $\mathbf{Facet}_j$ is denoted as $(\mathbf{FV}_{ij1}, \mathbf{FVW}_{ij1})$ $(\mathbf{FV}_{ij2}, \mathbf{FVW}_{ij2})$, …, *etc*. Each column in the table represents the facet-value pairs in one facet, and each row represents the facet descriptor associated with one component, i.e. all facet-value pairs for a component.

SRF classification process

SRF's classification process can be divided into three stages: *Acquisition*, *Analysis*, and *Clarification*. Each facet of a software component's description is acquired from the software library. These descriptions are analysed to extract *candidate pairs*. SRF clarifies candidate-pair words as follows: each candidate pair is considered a facet-value pair if listed in the hierarchical thesaurus or special-word dictionary; otherwise, it is redefined by domain experts to get facet-value pair(s). These facet-value pairs are collected as component facet descriptors and put in the classification catalog. Domain experts can also define facet-value pairs and put them in the catalog. The data flow diagram for SRF's classification process is shown in Figure 7.

Here the Smalltalk-80 library is used as an example. In Smalltalk-80, a method is treated as a software component for classification purposes. Three facets, *Function*, *Object*, and *System-Type*, are used to classify software components. A method is

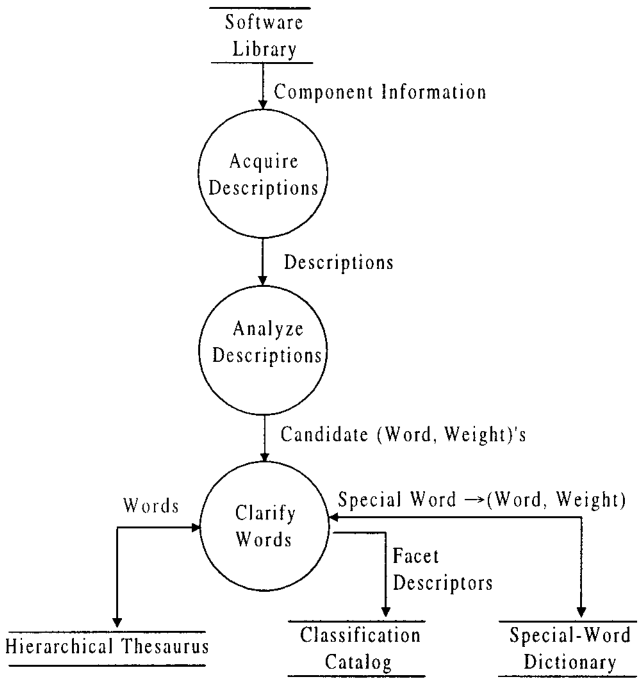


Figure 7. The SRF classification process data flow diagram

classified by analysing its execution functions, the objects it manipulates, and the functionally identifiable and application-independent modules it has. These facets are defined in Holm and Maarek,[9] and were selected because they are the most likely ones for users to consider during search processes. The detailed classification process for the Smalltalk-80 library is discussed below.

1. *Acquisition*. This step acquires software component descriptions. In the Smalltalk-80 library, the descriptions of three facets, *Function*, *Object*, and *System-Type*, are acquired from the names of methods, classes, and functional categories of classes, respectively. These names are used as descriptions because they represent software component indices in current libraries, and are used for automatic classification here. Table II contains some of the descriptions acquired from the Smalltalk-80 library.
2. *Analysis*. This step analyses the acquired descriptions to extract *candidate pairs*. The descriptions in Table II are simply formatted as follows: in System-Type facet, descriptions are sequences of words in which each pair of words is separated by a hyphen. In Object facet, descriptions are sequences of concatenated words, each beginning with capital letter. In Function facet, descriptions are the same as in Object facet except for the first word. Word extraction is done facet by facet, and each word extracted forms a candidate pair with a weight of 1. Some of the extracted candidate pairs are shown in Table III.
3. *Clarification*. This step clarifies candidate-pair words that are not in the hierarchical thesaurus or special-word dictionary. Here an example is used to show how clarification is accomplished. In Table III, three words, *sequenceable*, ‘,’, and ‘=’ need clarification. First, the standard forms of these words are recognized. The word *sequenceable* is recognized as *sequence*, and the facet-value pair is defined as (*sequence*, 1) instead of (*sequenceable*, 1). The word ‘,’ represents a special operation. Domain experts defined two pairs (*comma*, 1) and (*operator*, 1) as representing ‘,’. Correspondingly, the process inserts ‘,’ in the hierarchical thesaurus, and two statements ‘,’ $\longrightarrow$ (*comma*, 1) and ‘,’ $\longrightarrow$

Table II. Some acquired descriptions from the Smalltalk-80 library

System-Type Facet	*Object* Facet	*Function* Facet
Collections-Abstract	ArrayedCollection	add
Collections-Abstract	ArrayedCollection	defaultElement
Collections-Abstract	ArrayedCollection	storeElementsFrom
Collections-Abstract	ArrayedCollection	…
Collections-Abstract	SequenceableCollection	,
Collections-Abstract	SequenceableCollection	=
Collections-Abstract	SequenceableCollection	copyFrom
Collections-Abstract	SequenceableCollection	copyReplaceAll
Collections-Abstract	SequenceableCollection	copyReplaceFrom
Collections-Abstract	SequenceableCollection	…
Collections-Arrayed	Array	…
Collections-Arrayed	ByteArray	…
Collections-Arrayed	IntegerArray	…
…	…	…

Table III. Some extracted candidate pairs

System-Type Facet	*Object* Facet	*Function* Facet
(collection, 1), (abstract, 1)	(array, 1), (collection, 1)	(add, 1)
(collection, 1), (abstract,1)	(array, 1), (collection, 1)	(default, 1), (element, 1)
(collection, 1), (abstract, 1)	(array, 1), (collection, 1)	(store, 1), (element, 1), (from, 1)
…	…	…
(collection, 1), (abstract, 1)	(sequenceable, 1), (collection, 1)	(‘,’, 1)
(collection, 1), (abstract, 1)	(sequenceable, 1), (collection, 1)	(“=”, 1)
(collection, 1), (abstract, 1)	(sequenceable, 1), (collection, 1)	(copy, 1), (from, 1)
(collection, 1), (abstract, 1)	(sequenceable, 1), (collection, 1)	(copy, 1), (replace, 1), (all, 1)
(collection, 1), (abstract, 1)	(sequenceable, 1), (collection, 1)	(copy, 1), (replace, 1), (from, 1)
…	…	…

(*operator*, 1) in the special-word dictionary. Thus, three facet-value pairs (‘,’, 1), (*comma*, 1), and (*operator*, 1) were defined. The word ‘=’ also represents a special operation. ‘=’ was put into the hierarchical thesaurus directly, and two statements ‘=’ ⟶ (*equal*, 1), and ‘=’ ⟶ (*operator*, 1) were put into the special-word dictionary. Three facet-value pairs (‘=’, 1), (*equal*, 1), and (*operator*, 1) were then defined.

Advanced discussion

The classification process must be discussed in detail for general libraries. In the analysis step, acquired descriptions are either formatted or unformatted. Analysis of formatted descriptions, in the example above can be carried out easily. Unformatted descriptions are usually in natural language. Two approaches[6,7] have been proposed to extract candidate pairs under these circumstances. The GURU system[6] automatically extracts *lexical affinity* (**LA**) from natural-language documents. Each extracted **LA** contains two words and an indicator of importance (ρ) of the **LA** to a document. The **LA**’s can be used as candidate pairs. Using the same example used in Reference 5, the **LA**’s of the utility **mkdir** are shown in Table IV. They are automatically extracted from the **mkdir** document. If a word exists in more than one **LA**, the maximum ρ-value is used as the weight for the word. Here, the candidate pairs for **mkdir** are (*directory*, 5.08), (*make*, 5.08), (*create*, 2.74), (**mkdir**, 2.74), (*permission*, 1.48), and (*write*, 1.03).

The classification process may be semi-automatic, i.e. the first two steps can be

Table IV. **LA**’s for **mkdir** ranked by ρ-values

LA’s	ρ
directory make	5.08
create **mkdir**	2.74
directory **mkdir**	2.55
directory permission	1.48
directory write	1.03

automated to reduce the effort required to classify a software library. Intervention by domain experts is needed mainly during clarification.

The example in the preceding section indicated only a few clarification cases. A more general method for clarifying words can be defined as follows. A candidate pair **(WD, WT)** whose **WD** is in the hierarchical thesaurus is treated as a facet-value pair directly. If **WD** has **n** statements ($n \geq 1$) defined in the special-word dictionary, i.e.

$$\mathbf{WD} \rightarrow (\mathbf{CWD}_1, \mathbf{CWT}_1),\ \mathbf{WD} \rightarrow (\mathbf{CWD}_2, \mathbf{CWT}_2),\ \ldots,\ \mathbf{WD} \rightarrow (\mathbf{CWD}_n, \mathbf{CWT}_n)$$

the corresponding facet-value pairs are defined as follows:

$$(\mathbf{WD}, \mathbf{WT}),\ (\mathbf{CWD}_1, \mathbf{WT} \times \mathbf{CWT}_1),\ (\mathbf{CWD}_2, \mathbf{WT} \times \mathbf{CWT}_2),\ \ldots,\ (\mathbf{CWD}_n, \mathbf{WT} \times \mathbf{CWT}_n)$$

If **WD** or its standard form still cannot be found, domain experts select one (only) of the following.

1. **WD** represents a special operation. Domain experts define one or more pairs **(CWD, CWT)**'s to represent **WD**. Correspondingly, the process inserts **WD** in the hierarchical thesaurus and the statement(s) **WD** → **(CWD, CWT)** in the special-word dictionary. The rest of the facet-value pair construction is similar to the case in which **WD** has **n** statements in the special-word dictionary.
2. **WD** and a word in the hierarchical thesaurus have common concepts. **WD** is put in the hierarchical thesaurus and its ancestor concepts are those common ones. **(WD, WT)** is thus treated as a facet-value pair.
3. **WD** has no common concept with words in the hierarchical thesaurus. **WD** is put, with no ancestor concept, on Level 1 in the hierarchical thesaurus. **(WD, WT)** is treated as a facet-value pair. The software component represented by **(WD, WT)** can thus be returned when **WD** is in the user's query.
4. **WD** is useless. Its candidate pair is discarded.

THE SRF SEARCH PROCESS

After classification, the classification catalog of the software library has been constructed and users can now start SRF searches by entering queries and selecting proper search levels.

Suppose there are **n** facets $\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n$ to be classified and the selected search level is **SL**. A query **Q** contains a set of words. From a facet standpoint, **Q** can also be considered a set containing a word set for each of the **n** facets.

An SRF search process has four steps: *Query-Word validation*, *Query-Pair initialization*, *Synonymy generation*, and *Candidate-Component collection*. Below are presented the details of these steps with an example.

1. *Query-Word validation*. Let **W** be a query word, i.e. a word in the word set of **Q**, **W** may exist in the hierarchical thesaurus or in the special-word dictionary, or be unknown. Thus:
   (a) if **W** is found in the hierarchical thesaurus, **W** is valid;
   (b) if **W** is found in a statement **W** → **(CWD, CWT)** in the special-word dictionary, the **CWD** is valid;
   (c) if **W** is unknown, validation fails and SRF informs the user of this. Upon

receipt of such a message, users can continue searching by deleting **W**, or initiate a new search by modifying the current one.

SRF repeats these actions until all the query word(s) in **Q** have been validated, i.e. all have been found in the hierarchical thesaurus or the special-word dictionary. For a query {{*}, {*database*}, {*perform*, *sql*}} on **Level 2**, the asterisk '*' represents no restriction and no validation is necessary; *database* and *perform* are found in the hierarchical thesaurus; *sql* is found in the special-word dictionary. Thus, these three words are all valid.

2. *Query-Pair initialization.* All **Q**'s words have been found in the hierarchical thesaurus or special-word dictionary. If a query word **W** is found in the special-word dictionary, it will have one or more **(CWD, CWT)s** listed. These **CWDs** should be included in the query to find other related components. Instead of using query words directly, SRF defines a *query pair*, that contains a word and an associated weight to help the search. Each query word **W** is associated with a **QPS(W)**, a set of query pairs, for (**CWD, CWT)s** found in Step 1. In this step, SRF initializes a **QPS(W)** for each **W** in **Q** as follows:

   (a) **QPS(W)** = **{(W, 1)},** if **W** is in the hierarchical thesaurus.
   (b) **QPS(W)** = $\{(W, 1), (CWD_1, CWT_1), (CWD_2, CWT_2), \ldots, (CWD_k, CWT_k)\}$, if **W** has **k** statements $(k \geq 1)$, $W \rightarrow (CWD_1, CWT_1)$, $W \rightarrow (CWD_2, CWT_2)$, $\ldots$, $W \rightarrow (CWD_k, CWT_k)$, in the special-word dictionary.

Here **(W, 1)** is put in **QPS(W)** because the original word precisely expresses the user's meaning. SRF then defines each facet F_i in **Q** as a facet-query-pair set $FQPS(F_i)$, the union of **QPS(W)** for all F_i's words in **Q**. For **Q**, an integrated query **IQ** is a set of $FQPS(F_i)$ for all F_i in **Q**. For the query in Step 1, **QPS**(*database*) = {(*database*, 1)} and **QPS**(*perform*) = {(*perform*, 1)} since *database* and *perform* are found in the hierarchical thesaurus. **QPS**(*sql*) = {(*sql*, 1), (*structure*, 1), (*query*, 1), (*language*, 1)} since there are three statements, *sql* → (*structure*, 1), *sql* → (*query*, 1), and *sql* → (*language*, 1), in the special-word dictionary. The **IQ** equals {(*database*, 1), (*perform*, 1), (*sql*, 1), (*structure*, 1), (*query*, 1), (*language*, 1)}.

3. *Synonymy generation.* Let **(WD, WT)** be a query pair in $FQPS(F_i)$. In this step, SRF gets the synonymy set for **WD**, **SS(WD, SL)**, via synonymy generation, matches it with software components **SC**'s facet values, and then computes the closeness between **(WD, WT)** and each **SC** matched. An **SC** is a candidate when one synonym in **SS(WD, SL)** is a facet value of **SC** for facet F_i. The closeness between **(WD, WT)** and **SC** is called their *query-pair similarity*, denoted as **SimQP((WD, WT), SC, SL). SL** then becomes one of the three parameters because it affects the result **SS(WD, SL)** of synonymy generation and thus affects the query-pair similarity between **(WD, WT)** and **SC**. SRF repeats these processes until the synonymy generation for all query pairs in **IQ** has been completed and the query-pair similarities between query pairs and their candidate components have been computed. The computation of *query-pair similarity* is discussed below.

Figure 8 shows the synonymy generation for **WD**. **SS(WD, SL)** has **m** synonyms $(m \geq 1)$ that are also facet values in the facet descriptor for software component **SC**. Let these synonyms be denoted as $FV_1, FV_2, \ldots, FV_m$, respectively.

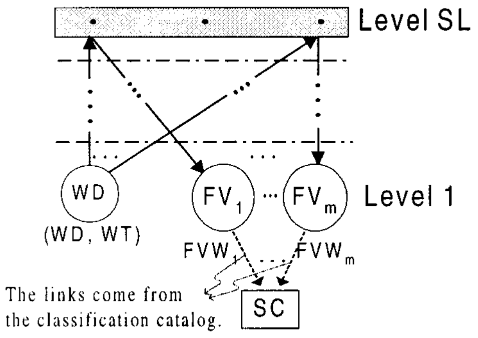


Figure 8. The synonymy generation of **WD**

The degree of closeness between **WD** and **FV$_j$** is defined as **SW(WD, FV$_j$, SL)**, the synonymity weight of **FV$_j$** resulting from **WD**. The computation of **SW(WD, FV$_j$, SL)** is done using Eq. (1). **FVW$_j$** also represents how well **FV$_j$** describes **SC**. The product of **SW(WD, FV$_j$, SL) × FVW$_j$** may thus represent the closeness between **WD** and **SC** via **FV$_j$**, and SRF defines the closeness between **WD** and **SC** as the maximum of the **m** products. **SimQP((WD, WT), SC, SL)** is computed as the product of the maximum and **WT**. The equation for **SimQP((WD, WT), SC, SL)** is shown below.

$$\mathbf{SimQP((WD, WT), SC, SL)} = \mathbf{WT} \times \max_{1 \leqslant j \leqslant m} [\mathbf{SW(WD, FV_j, SL)} \times \mathbf{FVW_j}]. \qquad (2)$$

For example, let one query pair of the **IQ** derived in Step 2 (*perform*, 1) own a possible candidate component *CDatabase::ExecuteSQL* whose facet descriptor is {{...}, {(*database*, 1)}, {(*execute*, 1)}, (*sql*, 1), (*structure*, 1), (*query*, 1), (*language*,)}. Figure 9 shows the synonymy generation for *perform*. *perform* has only one synonym, *execute*, in the facet descriptor for *CDatabase::ExecuteSQL*. *perform* has

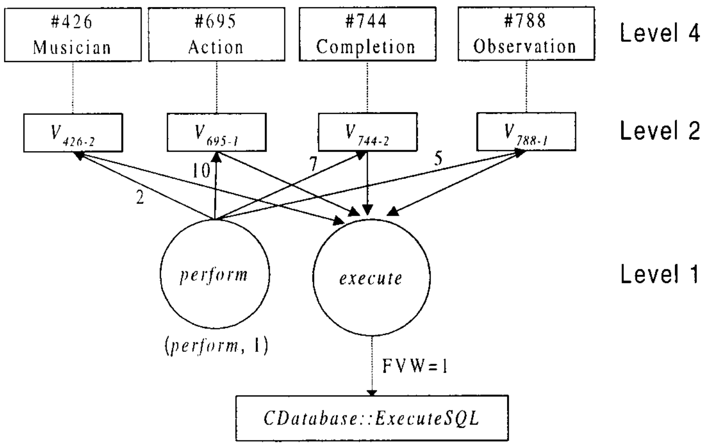


Figure 9. The synonymy generation for perform

four common concepts with *execute* on **Level 2**, and has thus four use frequencies, **UF**(*perform*, $V_{426\text{-}2}$), **UF**(*perform*, $V_{695\text{-}1}$), **UF**(*perform*, $V_{744\text{-}2}$), and **UF**(*perform*, $V_{788\text{-}1}$). Let the corresponding frequencies be 2, 10, 7, and 5 and labeled on the links from *perform* to the common concepts. The synonymity weight **SW**(*perform*, *execute*, **2)** equals 10 from Eq. (1). The query-pair similarity **SimQP**((*perform*, 1), *CDatabase::ExecuteSQL*, **2**) thus equals 10 from Eq. (2).

4. *Candidate-Component collection*. In this step, SRF collects the candidate components for query pairs in **IQ** and computes their similarity degrees, and puts the collected components in a *candidate-component set* **CCS(IQ, SL)**. Suppose there is a total of **k** query pairs in **IQ**, denoted as $(WD_1, WT_1), (WD_2, WT_2), \ldots, (WD_k, WT_k)$, the equation for **CCS(IQ, SL)** is shown below.

$$\mathbf{CCS(IQ, SL)} = \bigcup_{1 \leq i \leq k} \{\mathbf{sc} \mid \forall \mathbf{sc}, \mathbf{SimQP}((\mathbf{WD_i}, \mathbf{WT_i}), \mathbf{sc}, \mathbf{SL}) \neq \mathbf{0}\} \tag{3}$$

After **CCS(IQ, SL)** has been collected, their similarity degrees are also computed. SRF computes the similarity degree between **IQ** and **SC**, denoted as **Sim(IQ, SC, SL)**, by averaging all query pair similarities between the query pairs in **IQ** and **SC**. When an **IQ** has more query pairs with greater similarity degrees to **SC**, it should own a larger **Sim(IQ, SC, SL)**. Thus, the average equation for **Sim(IQ, SC, SL)** is defined as below

$$\mathbf{Sim(IQ, SC, SL)} = \frac{\mathbf{1}}{\mathbf{k}} \sum_{i=1}^{k} \mathbf{SimQP}((\mathbf{WD}_i, \mathbf{WT}_i), \mathbf{SC}, \mathbf{SL}) \tag{4}$$

Software component *CDatabase::ExecuteSQL* is used to demonstrate this step. The facet descriptor for *CDatabase::ExecuteSQL* is {{…}, {(*database*, 1)}, {(*execute*, 1), (*sql*, 1), (*structure*, 1), (*query*, 1), (*language*, 1)} and **IQ** equals {(*database*, 1), (*perform*, 1), (*sql, 1), (structure, 1), (query*, 1), (*language*, 1)}. Thus, the similarity degree **Sim(IQ,** *CDatabase::ExecuteSQL*, **2**) equals $\frac{\mathbf{1}}{\mathbf{6}}(\mathbf{5} \times \mathbf{MAX_{sw}} + \mathbf{10})$.

Finally, SRF returns the candidate components in order of their similarity degrees.

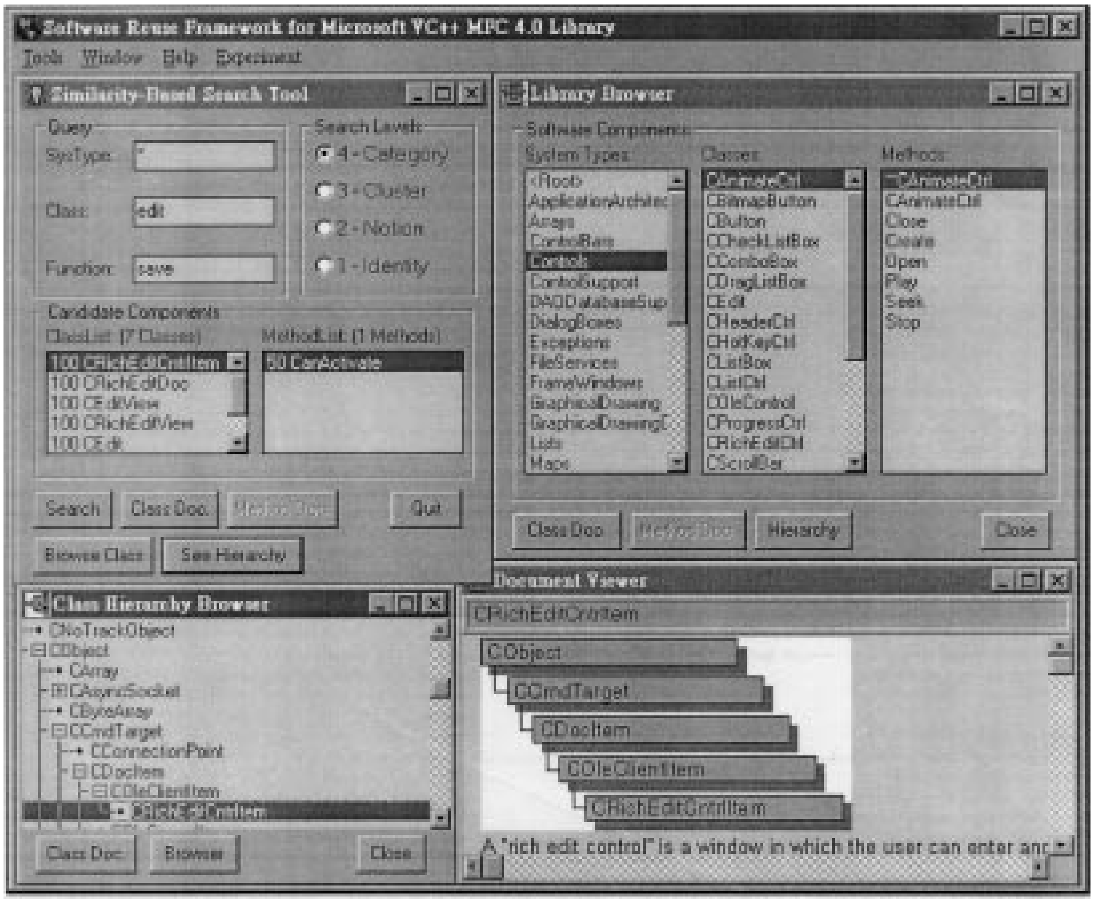
IMPLEMENTATION AND EXPERIMENTAL STUDIES OF SRF

The Prototypes

Two SRF prototypes have been implemented in Windows 95 using Visual Basic 3.0 and MS-Access 2.0. The hierarchical thesaurus was adapted from Roget13 Electronic Thesaurus, which contains 1022 categories and 17,359 words. Two sample object-oriented libraries, Smalltalk-80 (349 classes and 6697 methods) and MFC (Microsoft Foundation Classes: 210 classes and 4871 methods) libraries were used to demonstrate SRF's characteristics. Before SRF is implemented, $\mathbf{MAX_{SW}}$, a constant for computing the synonymity weight in Eq. (1), has to be determined. $\mathbf{MAX_{SW}}$ represents the maximum synonymity weight, a value larger than the use frequencies of all words in the hierarchical thesaurus. $\mathbf{MAX_{SW}}$ cannot reflect user's past experiences if it is set too high or low. For implementation convenience, *use times* are used instead of use frequencies. The use times of words were set to zero initially, and increased by one when the word occurred in a facet value of a selected SC during a query. In general, words in facet values of SCs but not considered meanings of those facet values are not often used. Thus, words whose use times increase rapidly are apparently used quite often, and may match user's ideas more closely. The use times of words that do not represent user's ideas increase slowly. In this implementation, $\mathtt{MAX}_{\mathtt{SW}}$ is defined as 100.

SRF provides six tools: a *classification tool*, a *hierarchical thesaurus examinator*, a *library browser*, a *class hierarchy browser*, a *document viewer*, and a *similarity-based search tool*. The first two are used by domain experts to classify software components. The others are used by general users to perform component searches in object-oriented libraries, where two popular browsers and a viewer are also commonly used in object-oriented programming environments. They were implemented to let users inspect the contents of classes or methods in SRF directly. The last one, the Similarity-Based Search Tool (SBST), is used to search components using techniques discussed in this paper.

Figure 10 shows the interfaces of these four tools in SRF. The upper right window



Query
SysType: *
Class: edit
Function: save

Search Levels: [✓ 4 - Category] 3 - Cluster 2 - Notion 1 - Identity

Candidate Components
ClassList (7 Classes): 100 CRichEditCntrItem, 100 CRichEditDoc, 100 CEditView, 100 CRichEditView, 100 CEdit
MethodList (1 Methods): 50 CanActivate

Software Components
System Types: <Root>, ApplicationArchitec, Arrays, ControlBars, Controls, ControlSupport, DAODatabaseSup, DialogBoxes, Exceptions, FileServices, FrameWindows, GraphicalDrawing, GraphicalDrawingC, Lists, Maps
Classes: CAnimateCtrl, CBitmapButton, CButton, CCheckListBox, CComboBox, CDragListBox, CEdit, CHeaderCtrl, CHotKeyCtrl, CListBox, CListCtrl, COleControl, CProgressCtrl, CRichEditCtrl, CScrollBar
Methods: ~CAnimateCtrl, CAnimateCtrl, Close, Create, Open, Play, Seek, Stop

CNoTrackObject
CObject
CArray
CAsyncSocket
CByteArray
CCmdTarget
CConnectionPoint
CDocItem
COleClientItem
CRichEditCntrItem

CRichEditCntrItem
CObject
CCmdTarget
CDocItem
COleClientItem
CRichEditCntrItem
A "rich edit control" is a window in which the user can enter anc

Figure 10. A sample user interface of SRF with MFC library

is the library browser. When a system type on the left is selected, the classes of the selected system type are shown in the middle list. Similarly, when a class is selected, its methods are shown in the right list. The lower right window is the document viewer, in which users can view documents of classes or methods by clicking the *Class Doc*. or *Method Doc*. button in the other tools. The lower left window is the class hierarchy browser. Users can click the '+' icon to expand or the '−' icon to collapse the subclasses of a class. A '◆' icon in front of a class means that the class has no subclass. The upper left window is the SBST. It includes three panels: *Query*, *Search Level*, and *Candidate Components*. Within the Query panel, on the top-left corner, users can enter their descriptions of desired components according to three facets: *System Type*, *Class* and *Function*. A facet description is either a set of words separated by commas, or an asterisk '*' representing no restriction. The Search Level panel on the top-right corner allows users to decide the level for search. Once users click the *Search* button, SRF performs a search based on the query entered, the selected search level, and the methods discussed in the section on 'The SRF Search Process'. SBST displays the results in a Candidate Components panel, which includes a *ClassList* of candidate classes and a *MethodList* of candidate methods in the selected class.

When classifying the Smalltalk-80 and MFC libraries, the first two steps, Acquisition and Analysis, automatically generated 1303 and 1033 distinct candidate words, respectively. In the Clarification step, 317 and 363 words were separately clarified by domain experts. The *clarification rate*, a numerical ratio of clarified words to candidate words, was computed for each facet and is shown in Table V. The clarification rates for the two libraries were 24 per cent and 35 per cent, i.e. one-fourth and about one-third of the candidate words needed clarification. This illustrates how the efforts required to classify the two sample libraries were greatly decreased through use of the hierarchical thesaurus.

In Table V, the total number of distinct words for Smalltalk-80 library is 1303 and the summation of that for each facet is 1575 (56+262+1257). The former value is less, because a candidate word may be used in two or three facets, and thus be counted more than once in the summation. This situation occurs to the number of clarified words.

Table V. The number of candidate words and clarified words for three facets

	System-type		Object		Function		Total number of distinct words	
	ST	MFC	ST	MFC	ST	MFC	ST	MFC
Number of candidate words	56	53	262	225	1257	1020	1303	1033
Number of clarified words	8	12	46	63	306	354	317	363
Clarification Rate (per cent)	14	23	18	28	24	35	24	35

(ST: Smalltalk-80 Library, MFC: Microsoft Foundation Class Library)

Table VI. Search time (in seconds) comparison between Groups I and II

Question no.	1	2	3	4	5	6	7	8	9	10	11	12	Average
Group I	118	219	186	174	196	260	310	230	639	418	587	990	361
Group II	114	206	172	128	116	226	184	208	617	161	160	957	271
Difference	4	13	14	46	80	34	126	22	22	257	427	33	90
Speedup (per cent)	3.5	6.3	8.1	35.9	69.0	15.0	68.5	10.6	3.6	159.6	266.9	3.4	33.2%

Experimental studies

Here an experiment was designed to evaluate the feasibility of SRF search process. It was carried out on SRF's version for the MFC library. Sixteen subjects (graduate students) participated in this experiment. They were divided into two groups: Group I subjects used three tools, the library browser, class hierarchy browser, and document viewer; Group II subjects used in addition the SBST. Twelve questions were designed, each containing statements describing a target component, which was either a class or a class and one of its methods, the subjects had to find. The subjects' actions in the experiment, such as starting searches, reading documents, etc., were automatically recorded in an action log file. After analysing the action log files for the sixteen subjects, two results were extracted:

1. *Search time*: this is the total time a subject took to find the target component in each question. Table VI lists the average search times for all subjects. Subjects in Group II, who also used SBST found the target components faster than those in Group I, and their use of SBST achieved 33.2 per cent speedup, on average.
2. *Tool use time*: this is the total time the subjects used tools for each question. Only the Group II subjects were calculated to determine whether they did use SBST in the experiment. Table VII lists the tool-use times for the three tools, SBST, Library Browser (LB), and Class Hierarchy Browser (CHB). Subjects obviously spent more time (32.2 per cent) using SBST than LB (9.7 per cent) and CHB (2.5 per cent). In total subjects spent 44.4 per cent of their time using tools, and 55.6 per cent reading documents. When the time spent for reading documents is ignored, percentages for use of the three tools can be

Table VII. Tool-use time (in seconds) for the three tools, similarity-based search tool (SBST), library browser (LB), and class hierarchy browser (CHB) with respect to total search time (TST)

Question no.	1	2	3	4	5	6	7	8	9	10	11	12	Sum	(per cent)
SBST	229	509	433	344	218	395	452	1180	1717	372	387	4012	10248	32.2
LB	31	121	91	55	180	250	67	29	763	166	100	1226	3079	9.7
CHB	23	24	23	0	24	6	60	3	239	8	0	357	797	2.5
TST	900	1765	1550	1022	1333	1676	1471	2979	4939	1342	1277	11613	31864	

calculated, and this is shown in Table VIII. Group II subjects spent almost three fourths of their total tool-use time on SBST, i.e. Group II subjects did use SBST in the experiment.

These experimental studies illustrate that SBST is useful in shortening search times. One thing that needs to be noticed is that the use frequencies of words representing concepts in the hierarchical thesaurus were all set to zero and weren't increased. The orderings of candidate components generated by SBST in this experiment might not be the best. Thus, search times may be even shorter than our experimental results after the use frequencies are increased from user feedback.

CONCLUSION AND FUTURE WORK

In this paper, we have presented in detail the Software Reuse Framework (SRF), which is based on a hierarchical thesaurus, and an experiment designed to evaluate SRF. The results of our experiment indicate that the SRF classification process is efficient, and that applying SRF does save search time.

Although SRF implementation and experimental studies look promising, at least three questions remain to be answered:

1. Is the model of four search levels suitable? This model was originally provided for users with different skill levels. Our experiment did not validate suitability.
2. How will use frequencies affect search time? In our experiment, we conjectured that the use of frequencies from long-term user feedback will save search time. This needs further study and evaluation.
3. Is SRF actually domain-independent? Our prototypes used Smalltalk-80 and MFC libraries, both are general-purpose code libraries. This is not sufficient to demonstrate domain-independence, due to lack of speciality.

Recently, Internet technology provides convenient ways for people to communicate with each other. It also provides convenient ways for servers to collect information from end-users. If SRF is extended on the Internet, it is easier to collect massive search information to evaluate the four search levels and use frequencies. The extension is currently being done with JAVA libraries. Besides, the Web information is diverse but not well-classified. The diverse Web information is useful to evaluate SRF's domain-independent characteristic. Tools based on SRF might be developed to assist users in searching Web information.

Table VIII. Tool-use time (in seconds) for the three tools, similarity-based search tool (SBST), library browser (LB), and class hierarchy browser (CHB) with respect to total tool use time (TTUT)

Question no.	1	2	3	4	5	6	7	8	9	10	11	12	Sum	(per cent)
SBST	229	509	433	344	218	395	452	1180	1717	372	397	4012	10248	72.6
LB	31	121	91	55	180	250	67	29	763	166	100	1226	3079	21.8
CHB	23	24	23	0	24	6	60	33	239	8	0	357	797	5.6
TTUT	283	654	547	399	422	561	579	1242	2719	546	487	5595	14124	

REFERENCES

1. T. C. Jones, ‘Reusability in programming: A survey of the state of the art’, *IEEE Transactions on Software Engineering*, **SE-10** (5), 488–493 (1984).
2. R. G. Lanergan and C. A. Grasso, ‘Software engineering with reusable designs and code’, *IEEE Transactions on Software Engineering,* **10** (5), 498–501 (1985).
3. C. W. Krueger, ‘Software reuse’, *ACM Computing Surveys,* **24** (2), 131–183 (1992).
4. T. Lsakowitz and R. J. Kauffman, ‘Supporting search for reusable software objects’, *IEEE Transactions on Software Engineering,* **22** (6), 407–423 (1996).
5. H. C. Liao and F. J. Wang, ‘Software reuse based on a large object-oriented library’, *ACM SIGSOFT Notes*, 74–80 (February 1993).
6. Y. Maarek, D. Berry and G. Kaiser, ‘An information retrieval approach for automatically constructing software libraries’, *IEEE Transactions on Software Engineering,* **17** (8), 800–813 (1991).
7. A. Smeaton, ‘SIMPR: Using natural language processing techniques for information retrieval’, *Proceedings of RIAO’91*, Huddersfield, April 1990; 152–160.
8. S. P. Arnold and S. L. Stepoway, ‘The REUSE System: Cataloging and retrieval of reusable software’, *Proceedings of COMPCONS ’87*; 376–379.
9. R. Helm and Y. S. Maarek, ‘Integrating information retrieval and domain specific approaches for browsing and retrieval in object-oriented class libraries’, *Proceedings of OOPSLA’91*, 47–61.
10. R. Prieto-Diaz and P. Freeman, ‘Classifying software for reusability’, *IEEE Software*, 6–16 (January 1987).
11. R. Prieto-Diaz, ‘Implementation faceted classification for software reuse’, *Communications of the ACM*, **34** (5), 89–97 (1991).
12. R. Prieto–Diaz and C. Braun, ‘Computing similarity in a reuse library system: An AI-based approach’, *ACM Transactions on Software Engineering and Methodology,* **1** (3), 205–228 (1992).