A Taxonomy for Identifying a Software Component from Uncertain and Partial Specifications

Giancarlo Succi, Francesco Baruchelli, Marco Ronchetti
LII/DISA, Università di Trento
Via Zeni 8, I-38068 Rovereto, Italia
e-mail: [First.Last]@lii.unitn.it

KEYWORDS: Uncertain Specifications, Reuse, Classification, Retrieval.

ABSTRACT

Software maintenance and reuse depends on a system for inserting and subsequently searching for software objects in a repository. A number of classical methodologies from library and information science exist and seem appropriate to this purpose. All of these methodologies bring to a high degree of uncertainty both in the specification of the searched objects and in the way an object is classified. To give a basis for a better understanding of these approximations, this paper presents a formal view of the basic library science methodologies for indexing and classification. The formalization permits the computational complexity of the different methodologies to be compared. In considering the special needs for extensions to the methodologies to support software reuse, a strategy based upon a hybridization of the simplest three methodologies is proposed rather than using or developing more complex methodologies.

The authors conclude that some extensions oriented towards the inevitable informal human side of the process may be necessary based on a domain analysis of the environment in which such methodologies must operate, but these extensions should be accounted for by engineering the proper elements into a formal framework which will support software reuse and reusability.

INTRODUCTION

Software reuse can help to increase the quality and to decrease the cost of a program, but it comes along with a fundamental problem: the location and the retrieval of software objects from a large collection called a repository. To reuse a software object, you first have to find it! The essence of the problem is to deposit software objects into the repository according to descriptions of the objects expressed in a formal language concise enough to serve as a subsequent indexing scheme. Then at some future time, a potential reuser of the object may describe his concept of a desired object in the same indexing scheme language.

The translation of an informal description into a limited formal language inevitably brings to a loss of information and to a certain degree of uncertainty in the characterization of the object.

The indexing scheme should be composed of two processes: the process in which a description is associated to the object which is inserted (classification), and the search query process (retrieval).

A high degree of uncertainty can be found in the retrieval process, too. In fact, the specifications of the objects to look for are very often uncertain and only partial: only very rarely one knows exactly what he is looking for, and even in this case the informal description which characterize the searched object has to be translated into the limited formal language, with a resulting increase in the degree of uncertainty of the query.

In order to better understand how this uncertainty affects the classification and the retrieval of software assets, it is important to deeply understand the commonly used classifi-
cation methodologies. For this reason in the following a formal representation of the methodologies will be given, along with an evaluation of their computational complexity. A survey of methods for representing software components for reuse was made by Frakes and Gandel in [8]. This survey includes traditional library science methods, knowledge-based methods and hypertextual systems, while completely formal approaches (such as predicate calculus, Horn clauses, or a declarative machine language) to represent search queries or classifications of inserted objects as formal specifications [2], [3], [5], [12], [13]), and automatic programming using generative methods [4], [9], [10], [11], [1]) have been excluded since they completely sidestep the classification/retrieval language compatibility issues and put other problems in their place. Artificial Intelligence and HyperText systems were included in this survey, but these systems work on a fine-grained level of detail requiring massive investments of effort to establish the complete connectivity of the elements in the cases of complex software objects/systems. It is commonly accepted as an alternative to formal approaches with a large-grained level of descriptive representation for software objects to use the widely known classification and indexing schemes originally developed in the context of library and information sciences. These classification and indexing methodologies, which will be intuitively summarized in the next section (A Taxonomy of the Common Classification Methodologies), have aspects that are easily formalized, as will be described in Section 3. Then the formal classification and indexing methodologies will be subjected to a subsequent complexity analysis, presented in Section 4, where we examine advantages in combining the various simple schemes over more complex variants. In Section 5, some possible extensions to make the simple schemes more operational in practical contexts will be covered, and in Section 6, we will summarize and draw conclusions.

A TAXONOMY OF THE COMMON CLASSIFICATION METHODOLOGIES

We present here a summary of the survey of Frakes and Gandel [8] to give an overview of the standard classification and indexing methodologies used by library and information science; further work by Frakes and other researchers in the field can be found in [7]. Traditional library and information science indicate with indexing, or classification, the process of creating a representation (essentially a description) of an object. Obviously there is a strong connection between classification and retrieval. A general retrieval process can be depicted as follows: the user of the information database created by a sequence of classifications and insertions decides what kind of component he wants, starting from his own informal description of the potentially interesting objects desired. From such an informal mental/verbal description, a formalized query is formulated which can be matched against the classification of existing components. The process presumes to have a repository filled with objects, with each object having been associated with a short abstract description serving to index it. This description has to be formalized into a classification scheme (analogously as is done for the query) so that queries can be successfully matched against it.

Many indexing languages have been developed for library and information retrieval applications; they define an item's physical location in a library and summarize what the item is about by describing its subject or content. What an information item considered to be 'about' may vary with the search situation which is part of the need for carefully engineering the classification and retrieval language so that it will indeed be useful.

Information may be attached to an object manually, from user input, or automatically, i.e., extracted from information such as text found in the object itself (presuming the object contains text itself) or documentation accompanying the object (such as a microfilm or an executable binary program).

![Figure 1: Taxonomy of Standard Vocabularies for Classification](image)

One can characterize such languages as either "languages with controlled vocabulary" or "languages with uncontrolled vocabulary" as shown in the above Figure 1. It must be noted that in the controlled indexing vocabularies, it is possible to limit some lexical aspect of the language (i.e., the terms or words in the language). In either controlled or uncontrolled, the rules for combining terms and the logical relationships are defined separately from the lexical primitives.

For controlled vocabularies, the essential idea is that by restricting indexers and searchers to a common set of terms, one a-priori encourages searches to match appropriate objects. Thus, the "languages with controlled vocabulary" were developed to help ensure that terms used by indexers and searchers would be the same. They can be separated into classed systems and keyword systems.
Classed Systems

In classed systems, the terms of the language are controlled and structured based on certain classes. All terms in the same class exhibit a similar set of properties that semantically justify their being assigned membership in that class. There are two types of classed systems, enumerated and faceted.

Enumerated Classification

In enumerated classification, a single subject matter class label must be assigned to an object. The system assumes that classes must be mutually exclusive, and classes cannot be combined to form new classes ex post facto (after the system has been devised) though classes may be later subdivided. This means that classes must have at least a partial hierarchical taxonomy. A simple example is the Dewey decimal system which has named classes assigned to decimal number where the highest level on classification is associated with the integer part and progressively smaller decimal fractions correspond to subdivisions of subject matter topics into subtopics.

The hierarchical structure makes it easy for searchers and indexers to interpret semantic relationships between subject matter terms. Users can easily modify their searches to be more specific or more general by moving up or down the classification hierarchy.

A disadvantage of enumerated methodology is that all possible class labels describing a domain are listed in an initial "classification scheme". Hence, the changes in such a scheme cannot be made without totally restructuring the system. In software reuse, i.e., applications where domains and terminology are constantly changing, this is a serious limitation.

Faceted Classification

In faceted classification, one or more descriptive properties or attributes of an object may be assigned a value from a pre-agreed structured set of basic terms associated with each such attribute. These attributes are limited in number and are called facets. Each facet conveys information about a particular descriptive aspect or property of a class or set of objects. The facets collectively may be thought of as spanning an attribute vector space which will be useful in describing any particular object or in describing groups (i.e., classes or sets) of objects according to logical conditions expressed over the facets: e.g., \[\text{[Functions=PopQueue]} \land \text{[ArgumentCounts>3]}\].

The structure of terms allowed within each facet reflects semantic relations between the terms - for instance, one term being more general than another or more specific than another or being a synonym or a mutually exclusive alternative for another term. The use of relations results from the need to reference objects which are entered and sought from varying views of specificity of the objects "aboutness" (what the object is about).

Since facets are similar to coordinates or bases for locating and describing objects, they are usually well defined (i.e., non-arbitrary and non-ambiguous) and as orthogonal to one another as possible. When searching for a particular object, one must specify terms for one or more relevant facets according to an idea of the class of objects that one may like to use.

Both searchers and indexers can benefit from the ability to describe sets or classes of objects using combinations of facet value expressions. This is particularly useful for indexers to easily assign composite classes during the indexing process, synthesizing new classes from the pre-existing classes that correspond to value assignments of single facets. Because of this flexibility, faceted system are also easier to update and modify as individual facets can be changed independently of other facets.

Unclassed or Keyword-Based Systems

Unclassed systems are more amorphous, less structured, than classed systems. Usually, they are open to the introduction of new concepts as new concepts evolve, whereas classed systems must rely on a fixed set of terms predefined by the class structure. Unclassed systems rely simply on making an assignment of symbols to objects from a fixed set of symbols known as keywords, and consequently they are commonly referred to as keyword based systems.

Keyword based approaches focus on attaching one or more keywords (acceptable natural-language words and phrases) to each object in order to describe its properties. Terms are typically arranged alphabetically.

However, keyword systems provide less information about the relationships between terms. Thus, there can be a mismatching of the objects being sought due to the lack of a common understanding between indexers and searchers regarding the semantics between different keywords. So, in order to provide both indexers and searchers information about the semantic relationships among terms as an alphabetically arranged keyword list, a thesaurus system is commonly included.

Much like classed systems, two possible approaches exist, corresponding to whether the keywords regard subject matter or an open variety of properties.

Subject-headings

Indexing languages as subject-heading systems does not allow the synthesis of basic terms to express composite concepts because all composite terms are created before the system is used by indexers. However, it does not preclude multiple keywords in a search that describes different views on the subject matter.
Some freedom is thus allowed to the searcher, and more
descriptive power is given to the indexer. Since there is no
assumed structure among the keywords, new terms can be
added at a later time - for instance when a completely new
technology is invented.

Descriptors

Descriptor systems use keywords designed to allow search-
ers to synthesize terms and composite terms using Boolean
operations. In this way, some of the syntactic finesse of fac-
eted approaches is given to a searcher though not to an
-indexer.
Keywords systems are easier to create and to modify than
classed systems. In fact they allow the addition of terms to
cover new concepts without affecting existing terms.

Free Text Retrieval

For uncontrolled vocabularies, the essential idea is that by
permitting flexibility both on the classification and retrieval
side, the language is a-posteriori adaptable to the needs.
The lack of restrictions on terminology allows indexers to
enter a more complete and characteristic description of an
object in the library/repository, and searchers can have
access to a wider set on concepts to match against - espe-
sially if the sought concept can be usefully be conceptual-
ized in more than one way. Thus, the "languages with uncon-
trolled vocabulary" were developed to help ensure that nat-
ural language and "free text" could be used to describe
objects with less cognitive overhead in classifying and
retrieving.

In an uncontrolled vocabulary, no restriction is placed on
which terms can be used to describe an item. Some subcate-
gories exist in the methodology, but all searches involve the
search for textual substrings within a textual description.
The source of the text may be extracted from a separate
description - e.g., an abstract or brief textual characteri-
ization associated with the item to be retrieved, or the text
may be derived directly from the object in the repository/library
itself. In the later case, the exact syntax or some reflection
of it (e.g., maintaining word orders of the main terms in a
phrase - also somewhat erroneously called "keyword in
context").

In addition, terms may have associated weights to show rela-
tive importance - either in queries or indexing entries.
These weights are usually derived from the frequency dis-
tribution of the terms within the objects. A free text based
solution can easily also allow Boolean combinations of
substrings in the textual description sought to be matched.
One significant advantage of free text is that since terms are
unrestricted and indexing terms can be made as specific as
it is possible to provide maximal descriptive power from a
human's point of view. On the other hand since such textual
descriptions are overly informal and tend toward natural
language, they are unfortunately ambiguous and prone to
mismatching or failure to match queries.

FORMALIZING THE METHODOLOGIES

This section is devoted to formally defining the different
classification mechanisms, the way the queries should be
posed for them and the connected answers.
For each mechanism, first the classification template is pre-
sented, then the query is formalized and the definition of
the result is stated; finally some examples are presented.

Keywords

The conceptually simplest mechanism for classifying and
retrieving objects is by means of a sequence of keywords.
Keywords represent a controlled vocabulary, and hence
constitute an explicitly defined keyword set:

\[ K = \{ k_1, k_2, k_3, ..., k_n \} \]

where \( k_1, ..., k_n \) are individual keywords. For the purpose
of classifying objects (denoted variously as \( O \), \( Q^k \), \( O^k \), ...
herein), a set of one or more keywords is associated to each
object so that a descriptor of an object can be formally
viewed as a non-empty subset of keywords:

\[ D^k(O) = \{ k \in K \} \subset K. \]

Moreover, the objects that we are interested in classifying
and retrieving based on keywords are collected into a set
called the Repository:

\[ \text{Repository}=\{ O_i \}. \]

For retrieval, a simple query is defined again (like a
descriptor) as a non-empty set of keywords:

\[ Q^k=\{ qk_i \} \subset K. \]

Usually, the keywords in a descriptor are considered as
constituting elements of a disjunctive assertion (connected
with an or operator), while the keywords in a query are
considered as forming a conjunctive condition (a test con-
structed with and operators). I.e.,

\[ D^k(O)=\forall qk_i \]

and

\[ Q^k=\forall qk_i. \]

Subsequently, the result of a query against the Repository
is

\[ R^k(Q^k) = \{ O \in \text{Repository}, \forall qk_i \in Q^k, qk_i \in D^k(O) \}. \]

This means that the result is the set of all objects with
descriptors containing all the keywords of the query.
For example, if we were dealing with microcomputers, we
might have keywords associated with various aspects of the
CPU, ROM memory, RAM memory, disk, type of display
and number of serial I/O ports. A descriptions of typical
system might look like:

\{ 1Mb_Ram, Color_Display, SCSI_disk, 68000_CPU \}

Note that not all relevant keywords have necessarily been
specified. This is not a requirement of the methodology.
A query might look like
\[ \text{[1Mb_Ram, Color_Display]} \]
This would certainly include the above descriptor as a result if it were recorded in the repository, but it could also produce
\[ \text{[1Mb_Ram, Color_Display, IDE_disk, 8080_CPU]} \]
as an additional result.
It is also possible to have more complex queries that contain or operators:
\[ Q^k_1 = Q^k_1 \vee Q^k_2 \vee \ldots \vee Q^k_m \]
where each subquery component \( Q^k_i \) is precisely as before, a list of keywords:
\[ Q^k_i = \{ q_{k_1}^i, q_{k_2}^i, \ldots, q_{k_{n_i}}^i \} \]
The result of such an or-query is the union of all the results obtained from each group of keywords in the or-query. Formally, according to the previously defined conventions, an or-query is a sequence of expressions composed of keywords partly connected with and and partly with ors:
\[ Q^k = (q_{k_1}^1 \land q_{k_2}^1 \land \ldots q_{k_{n_1}}^1) \vee (q_{k_1}^2 \land q_{k_2}^2 \land \ldots q_{k_{n_2}}^2) \vee \ldots \vee (q_{k_1}^m \land q_{k_2}^m \land \ldots q_{k_{n_m}}^m) \]
and the result of the query can be written as:
\[ R(Q^k) = R(Q^k_1) \cup R(Q^k_2) \cup \ldots \cup R(Q^k_m) \]
Returning to the earlier microcomputer repository, we see that an or-query might be expressed as
\[ \text{[1Mb_Ram, Color_Display]} \text{ OR } \text{[SCSI_disk, 68000_CPU]} \]
which could return a result
\[ \text{[1Mb_Ram, Color_Display, SCSI_disk, 68000_CPU]}, \text{[2Mb_Ram, B&W_Display, SCSI_disk, 68000_CPU]}, \text{[1Mb_Ram, Color_Display, 8000_CPU, Dinosaur]}, \text{[1Mb_Ram, Color_Display, IDE_disk, 8008_CPU]} \]

**Weighted Keywords**

Sometimes there are not any objects whose keywords match the query. In such cases, it can be interesting to determine the set of objects constituting "good" approximations of the desired object. To do so, besides the simple keywords methodology, it is possible to assign weights to keywords to determine their relevance to an object description or the query, i.e., the level of specificity the keywords have in describing an object. For instance, a set of keywords describing the pen I have on my desk is \{black ink, Parker, thin\}; JP is looking for a black ink Pelikan, and we do not have one; it is likely that the keyword Parker is less significant than the one black ink, therefore my Parker is likely to satisfy JP.

Weights can be considered as either properties of keywords or relevance to the larger context of the description or query, depending on the object described; we assume the latter case, since it is more general. Although weighted keywords per se are not one of the basic methodologies, they correspond to one form of the thesaurus which is a standard fixture of the keyword methodology. Moreover, weights can also be assigned to each keyword in a query or a description of an object.
The descriptor of an object can be synthesized as follows:
\[ D^w_k(O) = (\{d_{k_l}, dw_l\}) \]
that is, the descriptor is a set of couples formed by a keyword and a descriptor weight of such keyword. The query is again a non-empty set of keywords, now a query weight is associated with each keyword:
\[ Q^w_k = \{ (q_{k_j}, qw_{t_j}) \} \]
The result of a query is a function of the way a weight contributes towards scoring the match of an individual query keyword against a descriptor keyword in an object (a matching function \( \Phi^w_k \)) and of how such results are combined to determine the overall result for all the keywords in a query considered against all the keywords in a particular object (a cumulator function \( \Gamma^w_k \)); formally:
\[ R^w_k(Q^w_k, \Gamma^w_k, \Phi^w_k) = \{ O \in \text{Repository} : v^w_k(O) > v^w_k(O') \} \]
where the function \( v^w_k \) expresses distance or similarity between query and repository objects and is defined as:
\[ v^w_k(O) = \Gamma^w_k(\Phi^w_k(q_{k_j}, \text{matchwt}(q_{k_j}, D^w_k(O)))) \]
where \( \text{matchwt}(q_{k_j}, D^w_k(O)) = dw_l \) if \( q_{k_j} \in D^w_k(O) \); NULL otherwise.

Nearly always this equation is presented in its simplified form, where \( \Gamma^w_k \) is \( \Sigma \) (summation) and \( \Phi^w_k \) is \( \Pi \) (multiplication of weights); in such cases, the null value is 0.

Coming back to the earlier example, we can assume that a red Pelikan pen, and two black pens, a Parker and a Pilot, are available. We also assume that all the weights in the object descriptors are 1 and in the query the color weight is 1, while the weight for the brand is 0.5. Using the simplified view in executing JP's query for a black Pelikan, the values associated to the three pen objects will be 0.5 for the red Pelikan (1*0 for the color + 1*0.5 for the brand), 1 for the black Parker (1*1 for the color + 1*0 for the brand) and 1 for the black Pilot (same as the Parker). The result is therefore the set formed by the black Parker and the black Pilot since both of these object have the maximum value.

Using this approach, the answer may not be satisfied by the solution since it is an approximation. Furthermore, it is a delicate task to assign weights to keywords, both in the query side and in the description side.

Note also that this kind of search does not require any semantic inference on the keywords, it is just an optimization problem.

**Free Text**

Using a free text classification mechanism, the descriptor of an object is a sequence of natural language sentences informally describing the object. Informally, the descriptor of an object is represented as a sequence of words, quite like the descriptor used in the keyword based classification.
ever, different from the keyword case, the whole informal description is processed to determine the most relevant words of it, and these can be taken intact as subsequences with order dependent structure. In other words, the descriptor is not a set of terms, (in a set, the order of elements is undefined and no duplicates are allowed) but a list of descriptors, which can be further processed to determine the important words and the associated relevance. This can be formalized in the following way:

$$D_{inf}^R(O) = \{w_1, w_2, w_3, w_4, \ldots, w_n\}$$

and after the analysis, there exists a set of couples (describing word, relevance) as:

$$D^R(O) = \{(d_{wi}, r_{dwi})\}$$

A possible simple strategy to assign relevance values is to delete the conjunctions, prepositions, articles, pronouns, and other low importance words, and then tabulate the relative occurrence frequency of the remaining words. For instance, taking the IEEE definition of a process: a sequence of steps performed for a given purpose, initially the descriptor is:

$$D_{inf}^R(Process) = \{a, sequence, of, steps, performed, for, a, given, purpose\}$$

then the relevance is determined by eliminating the low importance words $a, of, for, and a, and then weighting each remaining word in terms of its relative number of occurrences. Here no object appears twice, so all are equally weighted and the final result is:

$$D^R(Process) = \{sequence, 0.2\}, \{steps, 0.2\}, \{performed, 0.2\}, \{given, 0.2\}, \{result, 0.2\}$$

More analysis can be applied to the words in order to eliminate further irrelevant terms (such as given) and possible synonyms. However such details are not analysed here since the focus is on the general taxonomy rather than on details.

A query is a set of words, quite like the case of keywords:

$$Q^R = \{qw\}$$

The result is the set of objects which maximizes a correlation value expressed in terms of relevances of the descriptors and a function, $\Gamma^R$, composing the relevances of each term in the query relative to the corresponding terms in the descriptor of an object from the Repository:

$$R^R(Q^R, \Gamma^R) = \{O: O \in \text{Repository}, \text{not } \exists \text{exists } O^k \in \text{Repository: } v^R(O^k) > v^R(O)\}$$

where the function $v^R$ is defined as:

$$v^R(O) = \Gamma^R(\{\text{match}(qw, D^R(O))\})$$

where match$(qw, D^R(O))$= $d_{wi}$ if $\exists$ exists $d_{wi}$, $qw$= $d_{wi}$ \wedge $(d_{wi}, r_{dwi}) \in D^R(O)$, $\quad$ otherwise.

For instance, again the function $\Gamma^R$ can be $\Sigma$ and other interpretations of the matching function may be possible. This method can be further refined adding weights to each word in the query. In this case, the query would be:

$$Q^R = \{(qw, r_{qw})\}$$

Therefore the result of such a query can be modelled in terms of an integrating function $\Gamma^R$ and a combining function $\Phi^R$:

$$R^R(Q^R, \Gamma^R, \Phi^R) = \{O: O \in \text{Repository, not } \exists O^k \in \text{Repository: } v^R(O^k) > v^R(O)\}$$

where the function $v^R$ is defined as:

$$v^R(O) = \Gamma^R(\{\text{match}(qw, r_{qw}, d_{wi}, D^R(O))\})$$

However the focus of this paper is more on the general taxonomy rather than on the details of how the different methods can be combined in all the possible ways; so we shall not dwell on such details much further.

Simple Faceted

With a simple faceted approach, the descriptor of an object is an attribute vector. The value of each component attribute of the vector has to be one from a set of its predefined values which are organized as a forest (a partial ordering corresponding to a group of trees). The forest defines a general to specific ordering of the values. Each arc in the tree is associated with a value ranking from 0 to 1 and defining the proximity of the derivation of the general to the specific relation between values: a value of 1 means equivalent whereas 0 means no arc (the values are not related at all nor are they mutually exclusive). A query is a template or partially instantiated vector of values. In the best case, an exact match exists; then the set of the exact matches is the result of the query. Otherwise, the result is the set of the best matches, defined in terms of the above-mentioned values of the arcs.

More formally, the attribute space is depicted as:

$$AS = \{V^k_1\} \times \{V^k_2\} \times \ldots \times \{V^k_n\}, \text{where} \quad \{V^k_i\} = \text{domain}(Facet_i)$$

and so the descriptor of an object can be written as a vector:

$$D^S(O) = (d^S_1, d^S_2, \ldots, d^S_n) \in AS, \text{such that } d^S_j \in \{V^k_j\}, \text{and the query correspondingly is a vector as well:}$$

$$Q^S = (q^S_1, q^S_2, \ldots, q^S_n) \in AS, \text{such that } q^S_j \in \{V^k_j\}.$$
where:

\[ P^{\text{sf}}(O^x, Q^o) = \text{IF} ((\Phi_{\text{sf}}(qv_i, dv_{v_i}))). \]

In the forest of values for each facet, a value or possibly two values can be associated to each arc specifying the proximity of its two nodes. Allowing two values permits modelling the unusual (but sometimes useful) case in which the upward value (i.e., from child to parent) is not equal to the downward value (i.e., from parent to child).

Such values are used by \( \Phi_{\text{sf}} \) to determine the proximity of the nodes corresponding to its two arguments—the facet value terms from a query and a repository object respectively. If they are neighbours, then the solution is straightforward since \( \Phi_{\text{sf}} \) takes the value of the connecting arc; otherwise, \( \Phi_{\text{sf}} \) searches for the shortest path between the two nodes, taking into account the general-to-specific taxonomy, i.e., computing the distance between the query-term node and the first ancestor common to both the query-term node and the repository-object-term node.

The function \( \text{IF} \) is used to integrate all the values coming from the different faces to determine the global proximity of two objects—the hypothetical query object and any repository object.

**Linear Simple Faceted**

The usual choice is to linearize the functions used to determine the proximity of two objects: to compute \( \Phi_{\text{sf}} \), all the values of the traversed links are multiplied. The values associated with the links span from 0 (no link) to 1 (complete equivalence). To compute \( \text{IF} \), the usual \( \Sigma \) (summation) over the values coming from the \( \Phi_{\text{sf}} \)s is taken.

As a side reference note that this can be formalized further using the common algebra vector notation: the face values can be viewed as a vector \( dv^f \) as well as the desired object \( qv^f \). Then \( p^f(O, Q^o) \) in the linear case can be represented as the usual scalar product: \( qv^f \cdot dv^f \), where the component by component multiplication is substituted by the \( \Phi_{\text{sf}} \).

In this case the result of a query can be expressed elegantly as:

\[ R^{\text{sf}}(dv^f) = \{O: O \in \text{Repository}, \not (\exists O^x \in \text{Repository}) \text{ where } (qv^o, dv^f \cdot qv^o) \}. \]

**Adding Weights to Faces**

Additionally, weights can be added to each face, both in the descriptor and in the query, to represent respectively the relevance of such face for the object and the interest of the user in such aspect of the object. In such a case, the descriptor of an object is represented as:

\[ D^{\text{wrf}}(O) = ((dv_{v_1}, r_{v_1}), (dv_{v_2}, r_{v_2}), \ldots, (dv_{v_n}, r_{v_n})). \]

and the query as:

\[ Q^{\text{wrf}}=((qv_{v_1}, r_{qv_1}), (qv_{v_2}, r_{qv_2}), \ldots, (qv_{v_n}, r_{qv_n})). \]

For computing the result, the weights must be taken into account. The \( r_{v_1} \) in the function \( \Phi^{\text{wrf}} \), and the \( r_{qv_1} \) in the \( \Gamma^{\text{wrf}} \), lead to the following result function:

\[ R^{\text{wrf}}(Q^{\text{wrf}}, \Gamma^{\text{wrf}}, \Phi^{\text{wrf}}) = \{O: O \in \text{Repository, not} (\exists O^x \in \text{Repository}) \text{ where } \}_{p^{\text{wrf}}(O^x, Q^{\text{wrf}}) > p^{\text{wrf}}(O, Q^{\text{wrf}})} \]

where:

\[ p^{\text{wrf}}(O, Q^{\text{wrf}}) = \text{IF} (\Phi^{\text{wrf}}(qv_{v_1}, dv_{v_1}, r_{v_1})). \]

And this can be linearized as:

\[ R^{\text{wrf}}(Q^{\text{wrf}}) = \{O: O \in \text{Repository, not} (\exists O^x \in \text{Repository}) \text{ where } p^{\text{wrf}}(O^x, Q^{\text{wrf}}) > p^{\text{wrf}}(O, Q^{\text{wrf}}) \}. \]

where:

\[ p^{\text{wrf}}(O, Q^{\text{wrf}}) = \Sigma (\Phi^{\text{wrf}}(qv_{v_1}, dv_{v_1}) \times r_{v_1} \times r_{qv_1}) \]

and \( \Phi^{\text{wrf}} \) is the usual maximum proximity function, the same as \( \Phi^{\text{sf}} \).

**Faceted with Semantic Connections**

In the case of a faceted description with semantic connections, common subtrees between elements of the same tree or of different trees can be present. Therefore, the domain of each face is organized as a directed graph. The formulae are formally still the same as in the case of the simple faceted classification mechanism (see Section Simple Faceted). Here below the corresponding forms are presented for the descriptor of an object, for the query and for the result:

\[ D^{\text{fac}}(O) = (dv_{v_1}, dv_{v_2}, \ldots, dv_{v_n}) \]

\[ Q^{\text{fac}}((qv_{v_1}, qv_{v_2}, \ldots, qv_{v_n}) \]

\[ R^{\text{fac}}(O^{\text{fac}}, \Gamma^{\text{fac}}, \Phi^{\text{fac}}) = \{O: O \in \text{Repository, not} (\exists O^x \in \text{Repository}) \text{ where } p^{\text{fac}}(O^x, Q^{\text{fac}}) > p^{\text{fac}}(O, Q^{\text{fac}}) \} \]

The difference is in the complexity analysis as will be shown in Section Complexity Analysis of the Methodologies.

**Linear and Weighted Faceted with Semantic Connections**

Furthermore, there can be a linearization of the classification and of the search leading to the following definitions:

\[ R^{\text{ifac}}(dv^f) = \{O: O \in \text{Repository, not} (\exists O^x \in \text{Repository}) \text{ where } (qv^o, dv^f \cdot qv^o, dv^f) \}

The difference between the \( \text{ifac} \) function we have here and the previous \( \text{ifac} \) is only in the computational overhead required by such a function. Section Complexity Analysis of the Methodologies is devoted to analysing such properties.

Obviously, weights can be added to each face and to each component of the query. In such a case we have the follow-
ing representation, which resembles very much that of the simple faceted representation (see Adding Weights to Faces); the only difference is in the complexity:

\[
D^{\text{wfac}}(O) = ((v_1, d_1), (v_2, d_2), \ldots, (v_n, d_n))
\]

\[
Q^{\text{wfac}}(O) = (q_1, r_1), (q_2, r_2), \ldots, (q_n, r_n)
\]

\[
R^{\text{wfac}}(Q^{\text{wfac}}, \Gamma^{\text{wfac}}, \Phi^{\text{wfac}}) = \{ (O, O^{\text{wfac}}) \in \text{Repository}, \not\exists O^x \in \text{Repository}: (O^{wfac}, Q^{wfac}, \rho^{wfac}) \}
\]

And this can be linearized as:

\[
R^{\text{wfac}}(Q^{\text{wfac}}) = \{ (O, O^{\text{wfac}}) \in \text{Repository}, \not\exists O^x \in \text{Repository}: (O^{wfac}, Q^{wfac}, \rho^{wfac}) \}
\]

\[
\rho^{\text{wfac}}(O, Q^{\text{wfac}}) = \Sigma(\Phi^{\text{wfac}}(q_i, d_i))
\]

where \(\Phi^{\text{wfac}}\) is the usual maximum proximity function, the same as \(\Phi^{\text{wsl}}\). As for the other cases of weighted representations, the details of these methodologies are not explained further in this paper, since they are not regarded as relevant for the present purposes.

**COMPLEXITY ANALYSIS OF THE METHODOLOGIES**

Here we make a summary of the computational complexity of the methodologies discussed in the previous sections. It is assumed that in general the repository is unsorted and that it has no more structure than a set. This reduces every case of a search to a sequential search through the repository. For the trees and graphs, we similarly assume that the structure is minimal - i.e., that only a set of nodes and a set of connecting arcs are known, so to locate an item requires a sequential search.

The parameters considered are the following:

- \(n_o\): the number of objects in the repository
- \(n_q\): the maximum cardinality of a query
- \(n_d\): the maximum cardinality of a descriptor
- \(n_f\): the maximum of the depths of the forests of the values of the faces
- \(n_n\): the maximum number of nodes in the graphs of the values of the faces
- \(n_i\): the number of faces

Table 1 at the end of this section gives the worst case computational overhead for each method previously discussed. In the following, the entries summarized in the table are to be verified. In all cases, we assume an unordered repository, so as to fairly compare the results of analysing the different methodologies.

**Keywords.** In the worst case, each keyword in the query has to be checked against each keyword of the object descriptor, leading to a complexity equal to \(\Theta(n_o \times n_q \times n_d)\).

**Weighted keywords.** In this case, the situation is the same as in the case of plain keywords, apart from the fact that the functions \(\Gamma^{\text{wkw}}(n_o)\) and \(\Phi^{\text{wkw}}\) have to be taken into account.

\(\Gamma^{\text{wkw}}(n_o)\) is computed at most \(n_o\) times and its complexity depends upon the maximum cardinality of the descriptors, \(n_d\). \(\Phi^{\text{wkw}}\) is computed at most \(n_q \times n_d\) times. The final formula is therefore \(\max(\Theta(n_q \times n_q \times n_d), n_q \times \Theta(\Phi^{\text{wkw}}, \Theta(\Gamma^{\text{wkw}}(n_o))))\).

**Linear weighted keywords.** When \(\Gamma^{\text{wkw}}(n_o)\) is \(\Sigma\), its associated complexity becomes \(\Theta(n_o \times n_q \times n_d)\), and when \(\Phi^{\text{wkw}}\) is the product, then its complexity is \(\Theta(n_o \times n_d)\). Substituting the \(\Gamma^{\text{wkw}}(n_o)\) and \(\Phi^{\text{wkw}}\) complexities into the above derived formula for weighted facets leads to a global complexity equal to \(\Theta(n_q \times n_q \times n_d)\). Note that this result is the same as the one obtained in the plain keywords mechanism; therefore the difference between the two is only in a constant factor. This is fairly obvious since the core of both methods requires matching elements from the domains of the keywords, the ones from the domains of objects found in the repositories against those from the query. The constant factor takes into account that the linear weighted case is more onerous to handle.

**Free text and linear free text.** These methods are analogous to keywords apart from having an infinite vocabulary. In particular free text resembles the weighted keywords, where the relevance of each word of the description is the weight of the keyword. Here there is no need for a \(\Phi\) function since combined effect of multiple query word matches is handled by \(\Gamma^{\text{ft}}\). The complexity is therefore \(\max(\Theta(n_o \times n_q \times n_d), \Theta(\Gamma^{\text{ft}}(n_o)))\). Since \(\Gamma^{\text{ft}}\) is \(\Sigma\) for the linear case, \(\Theta(\Gamma^{\text{ft}}(n_o)) = \Theta(n_o \times n_q \times n_d)\) and linear free text has consequently a complexity equal to \(\Theta(n_o \times n_q \times n_d)\).

**Simple faceted.** In the simple faceted mechanism, for each object in the repository the proximity of its facet values to the ones of the desired component has to be computed, which amounts to considering each face in turn. This operation depends on finding the distance between two nodes in the term space of a facet. Such distance is upper-bounded by the maximum depth of the corresponding tree of values which is upper-bounded by the maximum of such depths among all trees; therefore across all the faces, this collectively leads to a complexity of \(\Theta(n_o \times \Theta(\Phi^{\text{ft}}(n_o)))\). Finally, all the values coming from each face have to be combined by \(\Gamma^{\text{ft}}\), leading to a net result equal to \(\Theta(n_o \times \max(\Theta(\Gamma^{\text{ft}}(n_o)), \Theta(n_q \times \Theta(\Phi^{\text{ft}}(n_o))))))\).

**Linear simple faceted.** In this case, the complexity of \(\Phi^{\text{ft}}\)
is \( \Theta(n_g) \) since, intuitively, at most each of the two elements to be matched has to go to the root of the tree, and the complexity of \( \Gamma^{kbf} \) is straightforwardly \( n_g \) since \( \Gamma^{kbf} \) is the summation. Thus, substituting the \( \Phi^{kbf} \) and \( \Gamma^{kbf} \) complexities into the formula derived for simple facets, the result is \( \Theta(n_0 \times n_f \times n_g) \).

**Faceted with Semantic Connections.** The case of facets with semantic connections is entirely analogous to the simple faceted case, except for the fact that instead of \( n_g \), the global cardinality of the graph, \( n_g' \), has to be considered. Consequently, the complexity formula is \( \Theta(n_0 \times \max(\Theta(\Gamma^{fac}(n_f)), \Theta(n_f \times \Theta(\Phi^{fac}(n_g)))) \).

**Linear Faceted with Semantic Connections.** This case is again analogous to the linear simple faceted case, except for the fact that the minimum distance between two nodes of a graph (The above assumes the hypothesis of non-sparse graphs; otherwise if \( n_g \) is the number of arcs in the graph, the complexity is \( \Theta(n_g^2) \).) is \( \Theta(n_g^2) \). The resulting complexity is \( \Theta(n_0 \times n_f \times n_g^2) \).

### Table 1: Summary of the Computational Complexity of Different Classification Methodologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>keywords</td>
<td>( \Theta(n_0 \times n_q \times n_d) )</td>
</tr>
<tr>
<td>weighted keywords</td>
<td>( \max(\Theta(n_0 \times n_q \times n_d), n_q \times \Theta(\Phi^{wks}), \Theta(\Gamma^{wks}(n_q))) )</td>
</tr>
<tr>
<td>linear weighted keywords</td>
<td>( \Theta(n_0 \times n_q \times n_d) )</td>
</tr>
<tr>
<td>free text</td>
<td>( \max(\Theta(n_0 \times n_f \times n_q), \Theta(\Gamma^{f}(n_q))) )</td>
</tr>
<tr>
<td>linear free text</td>
<td>( \Theta(n_0 \times n_f \times n_q) )</td>
</tr>
<tr>
<td>simple facet</td>
<td>( \Theta(n_0 \times n_f \times n_q) \times \max(\Theta(\Gamma^{fac}(n_f)), \Theta(n_{f}\times\Theta(\Phi^{fac}(n_q)))) )</td>
</tr>
<tr>
<td>linear simple facet</td>
<td>( \Theta(n_0 \times n_f \times n_q) )</td>
</tr>
<tr>
<td>faceted with semantic</td>
<td>( \Theta(n_0 \times \max(\Theta(\Gamma^{fac}(n_f)), \Theta(n_f \times \Theta(\Phi^{fac}(n_g)))) )</td>
</tr>
<tr>
<td>connections</td>
<td>( \Theta(n_0 \times n_f \times n_g^2) )</td>
</tr>
<tr>
<td>linear faceted with</td>
<td>( \Theta(n_0 \times n_f \times n_g^2) )</td>
</tr>
<tr>
<td>semantic connections</td>
<td></td>
</tr>
</tbody>
</table>

**MINIMIZING THE SEARCH OVERHEAD**

There are definite savings to be gained in implementation which exploit additional structure imposed on the repository. For instance, \( \Phi^{fac} \) may be implemented as a large lookup table of node to node distances in which case its complexity is easily bounded by \( n_g^3 \). Similarly, minimally perfect hash functions or other rapid access techniques could be employed, but due to cost and other practical considerations associated with building the table, it can only be constructed periodically. This means that some degree of incremental sequential search is always necessary. The goal of these methodologies is to minimize search time and maximize the ease of use as well as the likelihood of identifying the right component. The most common approaches are in favour of weighted keyword based systems which are simple enough to be understood by novices, result in reasonably fast search and are quite effective in locating the desired artifacts within only a few interactions with the user [7], [6].

Another approach may be to first apply a simplified method retrieving a more general match than required. This would permit more involved retrieval to be applied selectively on another wave of processing that would restrict the consideration only to the objects retrieved on the first pass. In case of remote repositories, the situation is radically different since the network traffic has to be taken into account. Any network traffic overload is likely to largely overshadow the search overhead. This implies that a slower search mechanism is acceptable provided that it leads closer to the solution limiting as much as possible the interactions with the systems on the WAN.

Problems with distributed databases on wide area networks mean that it is important to work first in terms of duplicated distributed indexes rather than trying localize the complete repository. However, considerations of such concerns are beyond the scope of the present paper.

Thus, the strategy proposed from this consideration is that in the interest of keeping down the computational complexity one should allow a disjunctive hybridization of the simple approaches rather than extending these basic approaches to allow more sophistication (e.g., semantic connections on facets instead of remaining restricted to trees as the structure for facet values).

**CONCLUSIONS**

The paper began by considering the needs of a software classification (indexing) and retrieval (search) process with software reuse in mind. There is the informal aspect of the need for a language and understanding of semantics common to both sides of the process, based on the context in which the objects in a repository will be reused. To some extent, this can not be solved solely with a formalism. There needs to be agreement on the part of indexers and searchers about the "architectural style" used by a particular community of software producers. Employing indexers unfamiliar with the style of structuring programs used in
that community will virtually guarantee lack of reusability based on the classification process. For example, consider the two cases depicted in Figure 2 below: in one case the assumption is that a piece will be monolithic and in the other, the searcher structures the search with the hope of finding subcomponents satisfying separately formulated requirements. In the first case, there may be parts of the total requirements satisfying a series of partial requirements and their union would satisfy the total requirements collectively. Thus, the individual searcher must have a strategy that was foreseen by the indexer (or alternatively, the searcher must know the style used by the indexer) - otherwise, a query stands little chance of being stated in a way that is compatible with what has been inserted in the repository.

![Diagram](image)

Figure 2: Different Architectural Styles for Organizing the Same Functionality in a Software Product

This paper presented the basic library science approaches to indexing and classification and their implications for searchers of the repository. A formal view of the methodologies was put forward with the intent of comparing computational complexity of different methodologies. In considering the special needs for extensions to the methodologies to support software reuse, a strategy based upon a hybridization of the simplest methodologies was proposed rather than using or developing more complex methodologies.

The authors conclude that some extensions oriented towards the inevitable informal human side of the process may be necessary. This is the reason a domain analysis of the environment in which such methodologies must operate should be tightly coupled to the design of the termspace for keywords and facets. Any extensions should be accounted for by engineering the proper elements into a formal framework which will support software reuse and reusability, with the goal of avoiding undue computational complexity and recognition of the fact that ambiguity and mismatch of semantics prevent even the most sophisticated formalism from being effectively used. This is another argument in favour of simplicity rather than sophistication in classification and indexing schemes. If the scheme is well understood and computationally efficient, it will be successful; otherwise it will not.

REFERENCES


