A Formal View to Classification and Retrieval Mechanism for Reusable Objects

Giancarlo Succi, Carl Uhrik and Tullio Vernazza

DISA – Università di Trento
* DIST – Università di Genova

1 Introduction

One of the most fundamental problems in software maintainability and reusability regards locating and retrieving 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 indexing scheme should respect each of the two processes: the object insertion process, which is called classification, and the search query process, which is called retrieval. One or both processes generally involve human interpretations of intended informal meanings. Consequently, caution needs to be exercised to ensure that the formal language used to describe objects at the time of their insertion in the repository sufficiently matches the formal description that will be issued as a query by a potential reuser. This aspect of the reuse requires a cross-cultural fertilization of knowledge between indexers and eventual reusers which establishes a mutually beneficial understanding of the behavior patterns of the human elements in both processes.

Despite the inevitable informal aspect, the formal language used on each side of the classification-retrieval process can help channel the insertions and requests towards convergence when indeed there is a conceptual match. Such language engineering is in fact nothing new. Frakes and Gandel [8] give a survey of methods for representing software components for reuse which include traditional library science methods, knowledge based methods and hypertextual systems. Notably, completely formal approaches (predicate calculus, Horn clauses, or a declarative machine language) to represent search queries or classifications of inserted objects as formal specifications [4,5,6,10] and automatic programming using generative methods [2,3,7,9] have been excluded from that survey since these technics completely sidestep the classification/retrieval language compatibility issues and put other problems in their place. Although Artificial Intelligence and HyperText systems were included in the survey of Frakes and Gandel [8], 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. A commonly accepted alternative to formal approaches with a large-grained level of descriptive representation for software objects is to use the widely known classification and indexing schemes originally developed in the context of library and information sciences.

2 A taxonomy of the common classification methodologies

In the interest of giving an overview of the standard classification and indexing methodologies used by library and information science, we present here a summary of the survey of Frakes and Gandel [8].

In traditional library and information science, indexing, or classification, is the process of creating a representation which is essentially a description of an object.

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).

One can characterize such languages as either "languages with controlled vocabulary" or "languages with uncontrolled vocabulary". 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.

2.1 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.

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.

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.

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., $[\text{Function} = \text{PopQueue}] \land [\text{ArgumentCount} > 3]$.

2.2 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 pre-defined 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.

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

Descriptor systems use keywords designed to allow searchers to synthesize terms and composite terms using Boolean operations. In this way, some of the syntactic finesse of faceted 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.

2.3 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 - especially if the sought concept can usefully be conceptualized in more than one way. Thus, the "languages with uncontrolled vocabulary"
were developed to help ensure that natural 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 subcategories 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 characterization associated with the item to be retrieved, or the text may be derived directly from the object in the repository/library itself. In the latter 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").

3 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 presented, then the query is formalized and the definition of the result is stated; finally some examples are presented.

3.1 Keywords

The conceptually simplest mechanisms 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 \( O, O^2, O^3, ..., \) 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: \( D^k(O) = \{ d_k \} \subseteq K \).

Moreover, the objects that we are interested in classifying and retrieving based on keywords are collected into a set called the Repository: \(\text{Repository} = \{ O \} \).

For retrieval, a simple query is defined again (like a descriptor) as a non-empty set of keywords: \( Q^k = \{ q_k \} \subseteq K \). 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 constructed with and operators). I.e., \( D^k(O) \equiv \lor d_k \) and \( Q^k(O) \equiv \land q_k \).

Subsequently, the result of a query against the Repository is \( R^k(Q^k) = \{ O : O \in \text{Repository}, \forall q_k \in Q^k, q_k \in D^k(O) \} \).

This means that the result is the set of all objects with descriptors containing all the keywords in 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: \{1 Mb_Ram, Color_Display, SCS1_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: \{500k_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: \{1 Mb_Ram, Color_Display, IDE_disk, 8080_CPU\} as an additional result.

It is also possible to have more complex queries that contain or operators: \( Q^{ok} = Q_1^k \lor Q_2^k \lor \cdots \lor Q_m^k \), where each subquery component \( Q_i^k \) is precisely as before, a list of keywords: \( Q_i^k = \{ q_{ki,1}, q_{ki,2}, ..., q_{ki,n_i} \} \).

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 partly with ors: \( Q^{ok} = (q_{k1,1} \land q_{k1,2} \land \cdots q_{k1,n_1}) \lor (q_{k2,1} \land q_{k2,2} \land \cdots q_{k2,n_2}) \lor \cdots \lor (q_{km,1} \land q_{km,2} \land \cdots q_{km,n_m}) \). And the result of the query can be written as: \( R(Q^{ok}) = R(Q_1^k) \cup R(Q_2^k) \cup \cdots R(Q_m^k) \).

Returning to the earlier microcomputer repository, we see that an or-query might be expressed as: \{1 Mb_Ram, Color_Display\} OR \{SCS1_disk, 68000_CPU\} which could return as a result:

\{1 Mb_Ram, Color_Display, SCS1_disk, 68000_CPU\},
\{2 Mb_Ram, B&W_Display, SCS1_disk, 68000_CPU\}, \{1 Mb_Ram, Color_Display, 8000_CPU, Dinosaur\}, \{1 Mb_Ram, Color_Display, IDE_disk, 8008_CPU\}.

3.2 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: \(D^k_O = \{(dk_i, dw_i)\}\), 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: \(Q^k = \{(qk_j, qu_j)\}\).

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 \(\phi^k\)) 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 \(\Gamma^k\)); formally: \(\Gamma^k(Q^k, \Gamma^k, \phi^k) = \{O : O \in \text{Repository}, \not\exists O^x \in \text{Repository} : v^k(O) > v^k(O)\}\) where the function \(v^k\) expresses distance or similarity between query and repository objects and is defined as:

\[
v^k(O) = \Gamma^k(\{(\phi^k(qk_j, qu_j, matchut(qk_i, D^k_O)))\}),
\]

where \(matchut(qk_i, D^k_O) = \begin{cases} dw_i & \text{if } qk_i \in D^k_O; \\ \text{NULL} & \text{otherwise} \end{cases}\).

Nearly always this equation is presented in its simplified form, where \(\Gamma^k\) is \(\Sigma\) (summation) and \(\phi^k\) is \(\Pi\) (multiplication of weights); in such cases, the null value is 0. Note also that this kind of search does not require any semantic inference on the keywords, it is just an optimization problem.

### 3.3 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. However, 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, but a list of descriptors, which can be further processed to determine the important words and their relevance. This can be formalized in the following way: \(D^f_{\text{in}}(O) = \{w_1, w_2, w_3, w_4, \ldots w_n\}\).

After the analysis, there exists a set of couples (describing word, relevance) as: \(D^f_O = \{(dw_i, r_{dw_i})\}\).

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: \(a \text{ sequence of steps performed for a given purpose}\), initially the descriptor is: \(D^f_{\text{in}}(\text{Process}) = \{a, \text{sequence, of, steps, performed, for, a, given, purpose}\}\). then the relevance is determined by eliminating the low importance words \(a, of, for, and\), 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:

\(D^f(\text{Process}) = \{(\text{sequence, 0.2}), (\text{steps, 0.2}), (\text{performed, 0.2}), (\text{given, 0.2}), (\text{result, 0.2})\}\).

More analysis can be applied to the words in order to eliminate further irrelevant terms (such as \text{given}) and possible synonyms. However such details are not analyzed 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: \(Q^f = \{qw_j\}\). The result is the set of objects which maximizes a correlation value expressed in terms of relevances of the descriptors and a function, \(\Gamma^f\); composing the relevances of each term in the query relative to the corresponding terms in the descriptor of an object from the Repository: \(D^f(Q^f, \Gamma^f) = \{O : O \in \text{Repository}, \not\exists O^x \in \text{Repository} : v^f(O) > v^f(O)\}\) where the function \(v^f\) is defined as:

\[
v^f(O) = \Gamma^f(\{(\text{matchut}(qw_j, D^f(O)))\}),
\]

where \(\text{matchut}(qw_j, D^f(O)) = \begin{cases} r_{dw} & \text{if } \exists dw, qw_j = dw \land (dw, r_{dw}) \in D^f(O); \\ 0 & \text{otherwise}. \end{cases}\)

For instance, again the function \(\Gamma^f\) can be \(\Sigma\) and other interpretations of the matching function may be possible.

---

1. In a set, the order of elements is undefined and no duplicates are allowed.
This method can be further refined adding weights to each word in the query. In this case, the query would be: \( Q^{w_{i}} = \{(q_{w_{i}}, r_{w_{i}})\} \). Therefore the result of such a query can be modeled in terms of an integrating function \( \Gamma^{w_{i}} \) and a combining function \( \phi^{w_{i}} \): \( R^{w_{i}}(Q^{w_{i}}, \Gamma^{w_{i}}, \phi^{w_{i}}) = \{O : O \in \text{Repository}, \not\exists O^{x} \in \text{Repository} : v^{w_{i}}(O^{x}) > v^{w_{i}}(O)\} \) where the function \( v^{w_{i}} \) is defined as: \( v^{w_{i}}(O) = \Gamma^{w_{i}}(\phi^{w_{i}}(q_{w_{i}}, r_{w_{i}})) \).

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.

### 3.4 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 forest 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 abovementioned values of the arcs.

More formally, the attribute space is depicted as: \( AS = \{v_{1}^{f}\} \times \{v_{2}^{f}\} \times \cdots \times \{v_{n}^{f}\} \), where \( v_{i}^{f} \) = \( \text{domain(Facet}_{i}\) , and so the descriptor of an object can be written as a vector: \( D^{f}(O) = (dv_{1}^{f}, dv_{2}^{f}, \ldots, dv_{n}^{f}) \) \( \in AS \), such that \( dv_{i}^{f} \in \{v_{i}^{f}\} \), and the query correspondingly is a vector as well: \( Q^{f} = (qv_{1}, qv_{2}, \ldots, qv_{n}) \) \( \in AS \), such that \( qv_{i} \in \{v_{i}\} \).

The result of the query can be formally expressed as the set of the objects that are closest to the desired object, closest in terms of the values of the faces of their descriptors and in terms of two functions. One function, \( \phi^{f} \), determines the proximity of the value of each single face of all the objects in the repository to the corresponding facet value of the desired query object. The other function, \( \Gamma^{f} \), integrates the proximities of the different faces to determine the global proximity of each object in the repository to the desired object.

\( R^{f}(Q^{f}, \Gamma^{f}, \phi^{f}) = \{O : O \in \text{Repository}, \not\exists O^{x} \in \text{Repository} : p^{f}(O^{x}, Q^{f}) > p^{f}(O, Q^{f})\} \). \( p^{f}(O, Q^{f}) = \Gamma^{f}([\phi^{f}(qv_{i}, dv_{i})]) \).

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 modeling 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 \( \phi^{f} \) 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 neighbors, then the solution is straightforward since \( \phi^{f} \) takes the value of the connecting arc; otherwise, \( \phi^{f} \) 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 \( \Gamma^{f} \) 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.

### 3.5 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 formulas are formally still the same as in the case of the simple faceted classification mechanism (see Section 3.4). Here below the corresponding forms are presented for the descriptor of an object, for the query and for the result:

\[
D^{sc}(O) = (dv_{1}^{j}, dv_{2}^{j}, \ldots, dv_{n}^{j})
\]

\[
Q^{sc} = (qv_{1}, qv_{2}, \ldots, qv_{n})
\]

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

\[
p^{sc}(O, Q^{sc}) = \Gamma^{sc}(\phi^{sc}(qv_{i}, dv_{i}))
\]
4 Conclusions

The paper began by considering the needs of a software classification (indexing) and retrieval (search) process with software reuse in mind. The informal aspect of the common language and understanding of semantics 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. Introducing 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 for example the two cases depicted in 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 his total requirements satisfying his requirements partially and their union may appear to satisfy the total requirements superficially, but the individual searcher must have a strategy that is foreseen by the indexer - otherwise, he never states his query in a way that is compatible with what has been inserted in the repository.

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 based. This is the reason a domain analysis of the environment in which such methodologies must operate is tightly coupled to the design of the term space for keywords and facets.

References