Intelligent Classification and Retrieval of Software Components

Andreas S. Andreou, Dimitrios G. Vogiatzis, George A. Papadopoulos

Department of Computer Science
University of Cyprus
75 Kallipoleos Str, P.O.B. 537
CY-1678 Nicosia, Cyprus
E-mail: {aandreou, dimitrv, george}@cs.ucy.ac.cy

Abstract

This work proposes a new methodology for intelligent classification and retrieval of software components based on user–defined requirements. The classification scheme utilizes a dedicated genetic algorithm which evolves a small number of classifiers by dividing the set of available components stored in a database into certain subsets (clusters). Each classifier thus becomes the leader-representative of its cluster. When a user wishes to trace a component he/she identifies the desired characteristics (component profile) which are then compared with the characteristics of the available classifiers. The closest classifier matching the required characteristics over a user-defined threshold will result in the "winning" set of components belonging to its cluster, which are presented to the user in descending matching fitness. We have validated our methodology over a synthetic dataset of components and the results obtained were very encouraging. Last, we present the web application developed to support the proposed intelligent classification method.

1. Introduction

Software components may be conceived as independent and autonomous units which are offered by vendors and can be "glued" together, via their public interfaces, to form a complete software project, which is more reliable than development *ex nihilo* [10].

To this end, users of software components have to search between the components available in the market, evaluate their characteristics and purchase the most suitable ones for integration according to the functional requirements and constraints of their project. However, currently the available schemes in the market for searching and retrieving components are insufficient in terms of: a rich set of characteristics that would allow locating components using diversified components properties and features with the ability of combinations, a fast and robust retrieval scheme, and an efficient mapping of user requirements onto component characteristics.

We propose an innovative way to classify components with the aim of offering fast retrieval of the correct components according to the preferences of the user. In our proposition, component classification is based on a predefined set of characteristics, from which the user may select those of interest (preferences). The user's preferences are utilized by an intelligent algorithm to return those software components the characteristics of which match the preferences up to a user defined level.

The rest of the paper is organised as follows: Section 2 presents a short literature review. Section 3 presents the proposed method. Section 4 presents a set of experiments, followed by a brief presentation of the supporting software tool. Finally, Section 6 draws the conclusions and suggests future research steps.

2. Relevant Work

A fundamental problem for software component providers is that of organising a collection of components for fast retrieval. Computational Intelligence (CI) technologies such as evolutionary computation, neural networks, clustering algorithms [8], and fuzzy sets [2,9] have recently been used in component based software engineering [6].

There are two popular representation schemes for software components. The first is based on a controlled vocabulary, which is organised in a strict hierarchical way. This scheme has received a lot of criticism as being inflexible and not producing a good classification [4]. The second approach postulates a representation which is based on a series of characteristics (attributes) that assume values. This scheme is more flexible with regards to expansibility [4], which we have also adopted in the current work.

One of the earliest efforts in classification of software reusable components is based on a faceted scheme [3]. Another influencing component selection system has been developed in the context of the CdCE project [7]. Finally in the CLARiFi project a dynamic classification schema has been adopted [1].

3. An Intelligent Method for Classification and Retrieval of Software Components

Our method performs intelligent components classification and fast retrieval of a requested component via a web accessible software tool. The classification/retrieval procedures are based on a dedicated genetic algorithm that processes a set of predefined component characteristics. In what follows, we describe the basic concepts of our method:

Encoding: An encoding scheme for the software component characteristics maps their initial form (whatever that may be, e.g. linguistic terms, numerical types, etc.) to a form that will be used by a genetic algorithm to discover component classifiers. In our case we used binary strings (series of 0/1) to encode the component characteristics, the length of which depends on the aggregation of the number of bits needed to represent all possible values of each distinct characteristic.

Classifier Discovery: The genetic algorithm attempts to discover several different classifiers, each of which classifies a number of software components into a homogenous set in terms of characteristics. The classifying sets may have common elements as the classification process is based on component characteristics, with which it attempts to find large groups of components with common values. Typically, there will be a large number of components classified against a small number of classifiers (20 in our case). Thus, searching for a component will be performed by examining the user preferences against the classifiers rather than the actual components, something which will result in a fast process. It is also possible that some components may not be classified at all as this depends on the selection of a threshold parameter that specifies the similarity of a component with a classifier (i.e. the number of perfectly matched characteristics).

Component Retrieval: This is the situation where a user tries to locate a specific component. First, she/he determines the desired values of the component characteristics, thus forming her/his preferences. Second, she/he sets the matching threshold value (obviously the lower the threshold value the more components will be returned). The system will then encode the user's request as a bit string and will compare it against all classifiers. The closest match will signify the "winning" classifier and will trigger the return of those components that correspond to this classifier. An overview of the retrieval process is presented in
Figure 1, where the sets of classified components are represented as ellipses. The classifier corresponding to each class of components is drawn next to its set. The classes can be overlapping as previously mentioned. Moreover, we can observe a pool of unclassified components that may possibly occur as a result of the threshold set.

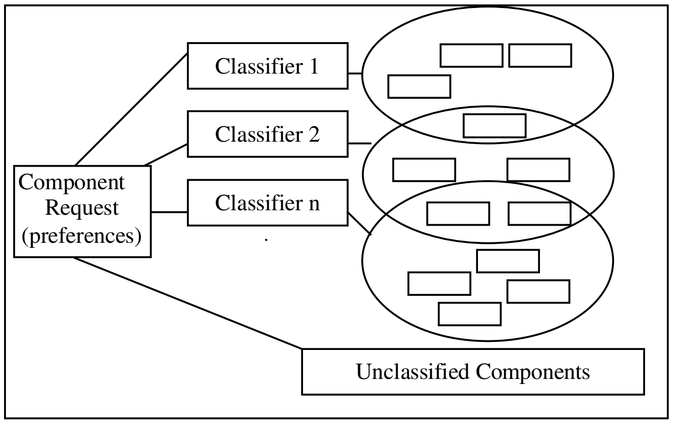


Figure 1. Overview of the system

3.1 Components characteristics and encoding

In the current experimental setting each component is described with a set of 15 characteristics, which were identified by examining a component from different perspectives (functional and non-functional). Our selected characteristics are: *General functionality*, *Specific functionality*, *Platform*, *Implementation language*, *Operating system independence, Synchronisation, Visibility of implementation*, *Price, Processor utilisation*, *Memory utilisation*, *Disk Utilisation*, *Binding*, *Data encryption*, *Data open format compatibility*. Each of the aforementioned characteristics is encoded as a string of 0/1. The encoding of a component is essentially the series connection of its characteristics' bit strings. We calculated the required bits for all characteristics and concluded to a string length equal to 60 bits. Figure 2 shows an example of a component with specific characteristics and its relevant bit encoding at the top.

111011001000011011100110001000001100010100010001001000101001

General functionality: Security
Specific functionality: Secure Email
Operating System Independence: Windows 98, 2000, Me, NT, XP, Unix, Linux
Implementation language: C#
Platform: Visual Studio .NET
Processor Utilization: 100-200MHz
Memory Utilization: 128MB
Disk Utilization: 100-500KB
Data Encryption: No
Data Open Format Compatibility: No
Password Protected: Yes
Visibility of implementation: Glass box
Price: $1-100
Synchronization: synchronous
Binding: Static

Figure 2. Example of component encoding

3.2 A Genetic Algorithm for identifying the classifiers

A dedicated GA [5] was developed to evolve candidate classifiers and select the optimal solution in terms of number of components in the corresponding classes, which works in discrete steps as follows:

1. Create a random population of 100 chromosomes - potential classifiers
2. For every generation of the genetic algorithm:
   2.1 Apply crossover to every pair of classifiers, where each pair is randomly selected according to a crossover probability
   2.2 Apply mutation to a randomly selected classifier according to a mutation probability
3. Perform component classification: for each of the 100 classifiers:
   a. Compare each classifier's values of characteristics with those of each component. If a component is close enough (determined by a threshold) to a classifier then assign the component to the class represented by this classifier. Normally, a large number of components will be assigned to each chromosome-classifier
   b. Select the top 20 classifiers (chromosomes) in terms of the number of assigned components. Then find the average number of assigned components of the 20 classifiers. This is the *average fitness* of the current generation
   c. If the average fitness of the current generation is greater than that of the previous generation then create a new population by selecting chromosomes according to their fitness and repeat from step 3. Otherwise do not create a new population and repeat from step 2

The above algorithm is repeated until a *termination condition* is reached. In our case the algorithm terminates if no improvement in the average fitness of the population is observed for 100 generations. A very important parameter is the value of the threshold, which determines whether a component belongs to a certain classifier. For example, a value of 40% means that at least 40% of the values of the classifier characteristics are identical to those of a component. This threshold essentially determines the "success" level of a classifier to gather a rich number of components in his class.

4. Experiments and Results

The first phase of the experiments was concerned with the *classification* of a pool of components, whereas in the second phase we investigated the *retrieval* of specific components.

For the *classification phase,* we created randomly 1000 components, each comprising 60 bits. The results reported are averages over 100 runs. The classification of the components is based on the 15 characteristics described in section 3.1. The threshold parameter is of paramount importance to our method, since it is a measure of similarity between the component characteristics and the classifier characteristics. We set the threshold value to assume the values of 30%, 40%, 50%, 60%, 70% and 80% for comparison purposes. The value of 30% produced classifiers, where each classified almost all of the available software components. This denotes that the classifiers derived cannot differentiate between the components. The threshold value of 80% did not produce good results either, because each classifier classified only between one and three components, which is also undesirable as it leaves many components unclassified (recall that there are only 20 classifiers). Similarly, unsatisfactory were the results when the threshold was set to 70%.

The results for the threshold values of 40%, 50% and 60% are listed in Table 2. The "Average" columns denote the average number of components classified by each classifier, while the "Not classified", denotes the number of unclassified components. The scores for 50% are quite successful, since there are no unclassified components and each classifier includes almost half of the components (47.5%). Thus, in the retrieval phase only half of the components need to be searched. Moving along the same lines, the value of 60% is also satisfactory since each class contains a small number of components (58.3 on average), but there is a significant number of unclassified components. The threshold of 40% is the worse since almost all of the components are classified by each classifier. This threshold, however, was also included in the retrieval phase for testing purposes. Furthermore, we have conducted experiments with 5,10 and 15 classifiers but the results were not satisfactory.

Table 2. Experimental result of component classification by 20 classifiers

Threshold value: 40%		*Threshold value: 50%*		*Threshold value 60%*	
Average	*Not classified*	*Average*	*Not classified*	*Average*	*Not classified*
937.03	0	475.99	0	58.30	312.87

To test the *retrieval* phase we created 10 random user requests searching for software components. Then we set the threshold from 40% to 70% at increments of 10% as shown in Table 3. We can observe that the 40% threshold returned a richer number of components, but not all of them were relevant to the users' query as expected. The 50% and 60% values retrieved less but more relevant components. The 70% threshold returned results for some of the queries only. However, the few cases for which we obtained results were highly relevant to the users' requests.

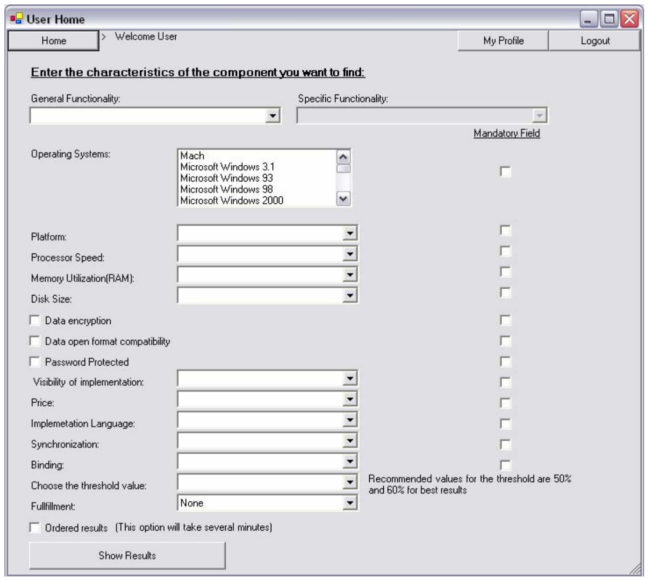
Table 3. Retrieval Phase Results

40%	*50%*	*60%*	*70%*
Average number of components per classifier			
31,9	23	8,2	2,4

The experiments were conducted with a tool that exists both as standalone system and as a web application. The main functions of the tool are summarized to searching for software components, inserting a new component in the database and some administration functions. The tool differentiates between simple users (reusers) who can only search for components and producers (component developers) who can insert new components. In Figure 3 we can see the characteristics that may be selected when searching for a component.

5. Conclusions and Future Work

We have designed and implemented an intelligent system for software component classification and retrieval. Classification is based on a small set of classifiers which are evolved (formed) with the aid of a dedicated genetic algorithm. Each classifier evolved by the GA attempts to classify the largest possible number of software components according to common characteristics. Retrieval of the relevant components may then be performed by comparing the developer's requirements with those of the classifiers. If the requirements are close enough to those of a classifier then the components that correspond to that classifier are returned. Thus, comparing a component's specifications with only those of the classifiers instead of the entire set of available components saves a significant amount of time and effort in the retrieval phase. A threshold is also used when evolving the classifiers, which determines which is the degree (percentage) of similarity with a classifier that is required to classify a component in a certain class. The threshold value has been found to have a profound influence in both the classifier's design phase (with the GAs) and the retrieval phase. The experiments revealed that the optimal threshold values for similarity were lying between 40% and 60%. In the future we intend to investigate the optimal action when a new component becomes available. There are two possibilities: either to find the closest matching classifier and to add it to the relevant class or to completely reorganise the components by running the genetic algorithm once more. Also, ranking the components that are returned by the system would be useful, (recall that the system returns all the components that correspond to a classifier). Such a ranking scheme could be based on how close a component is to the classifier or on user specified criteria which put bias on some characteristics (e.g. memory utilisation is more important that price). Another direction for enhancing our work will be based on a semantic interpretation of the user's/developer's queries against a repository which is inspired from the work presented in [11].



Enter the characteristics of the component you want to find:

Recommended values for the threshold are 50% and 60% for best results

Fullfillment: None

(This option will take several minutes)

Figure 3. Searching for a component

6. References

[1] CLARiFi, IST-1999-11631, http://clarify.eng.it

[2] E. Damiani and M. G. Fugini, "Automatic thesaurus construction supporting fuzzy retrieval of reusable components", Proceedings of the ACM Symposium on Applied Computing Tennessee, US, pp. 542-547, 1995

[3] R. P. Diaz, "Implementing Faceted Classification for Software Reuse", Communications of the ACM, Vol. 34, No. 5, pp.88-97, 1991

[4] W. B. Frakes and T. P. Pole "An Empirical Study Representation Methods for Reusable Software Components", IEEE Transactions on Software Engineering archive, vol. 20, no. 8, pp. 617-630, 1994

[5] D. E. Goldberg, "Genetic Algorithms", Addison-Wesley, 1989

[6] J. Lee, "Software Engineering with Computational Intelligence", Springer, 2003

[7] V. Maxville, J. Armarego, C.P. Lam, "Intelligent Component Selection", 28th Annual International Computer Software and Applications Conference, COMPSAC, pp 244-249, 2004

[8] S. Nakkrasae, P. Sophatsathit and W. R. Edwards, "Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification", International Journal of Computer & Information Science, vol 5, no. 1, March 2005

[9] W. Pedrycz and J. Waletzky, "Fuzzy clustering in software reusability", Software – Practice and Experience, vol.27, no.3, pp.245-270, 1997

[10] C. Szyperski, D. Gruntz and S. Murer, "Component Software", Addison Weslsey, 2002

[11] H. Yao and L. Etzkorn, "Towards a semantic-based approach for software reusable component classification and retrieval", Proceedings of the 42nd annual Southeast regional conference, pp. 110-115, 2004